Lecture Notes in Artificial Intelligence 2479

Subseries of Lecture Notes in Computer Science
Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Matthias Jarke Jana Koehler
Gerhard Lakemeyer (Eds.)

KI 2002: Advances in Artificial Intelligence

25th Annual German Conference on AI, KI 2002
Aachen, Germany, September 16-20, 2002
Proceedings

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Matthias Jarke
Gerhard Lakemeyer
RWTH Aachen, Informatik V
Ahornstraße 55, 52056 Aachen, Germany
E-mail: {jarke, lakemeyer}@cs.rwth-aachen.de

Jana Koehler
IBM Research Laboratory, Computer Science Research
Säumerstraße 4, 8803 Rüschlikon, Switzerland
E-mail: koe@zurich.ibm.com

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Advances in artificial intelligence : proceedings / KI 2002, 25th Annual German Conference on AI, KI 2002, Aachen, Germany, September 16 - 20, 2002. Matthias Jarke ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Lecture notes in computer science ; 2479 : Lecture notes in artificial intelligence)
ISBN 3-540-44185-9

CR Subject Classification (1998): I.2

ISSN 0302-9743
ISBN 3-540-44185-9 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York,
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna e.K.
Printed on acid-free paper SPIN: 10871403 06/3142 5 4 3 2 1 0

Preface

This year marks the 25th anniversary of the *Annual German Conference on Artificial Intelligence*. When starting in 1975 as the *German Workshop on AI (GWAI)* in Bonn, the meetings were primarily intended as a forum for the German AI community and thus, over the years, also reflected the development of AI research in Germany. Right from the beginning, the goal of the meetings has been to bring together AI researchers working in academia and industry to share their research results and interests in the field of artificial intelligence. In 1993, the name of the meeting changed to its current form *Annual German Conference on Artificial Intelligence*, or KI for short. Since KI-94 in Saarbrücken presentations have been in English and the proceedings have been published in the Springer LNAI Series. With that the meeting has become a truly international event, and still is in 2002.

This volume contains the proceedings of the *25th Annual German Conference on Artificial Intelligence*. For the technical program we had 58 submissions from 17 countries and from all continents except Australia. Out of these contributions slightly less than 30% (20 papers in total) were selected for presentation at the conference and for inclusion in the proceedings.

The contributions in this volume reflect the richness and diversity of artificial intelligence research. They cover important areas such as multi-agent systems, machine learning, natural language processing, constraint reasoning, knowledge representation, planning, and temporal reasoning. The paper

> "On the problem of computing small representations of least common subsumers" by Franz Baader and Anni-Yasmin Turhan (Dresden University of Technology)

stood out for its exceptional quality and the program committee selected it for the Springer Best Paper Award. Congratulations to the authors for their excellent contribution.

In addition to the technical papers, this volume contains the abstracts of the three invited presentations of the conference:

- Elisabeth André (University of Augsburg): *From Simulated Dialogues to Interactive Performances with Virtual Actors.*

- Michael Wooldridge (University of Liverpool): *Time, Knowledge, and Cooperation: Alternating-Time Temporal Epistemic Logic and Its Applications.*

- Dieter Fensel (Vrije Universiteit Amsterdam): *Semantic Web Enabled Web Services.*

Lastly, the volume features summaries of the three DFG Priority Programs on agent technology, also presented at the conference. These, together with their respective speakers, are:

- Thomas Malsch (Hamburg-Harburg University of Technology): *Socionics*
- Stefan Kirn (Ilmenau University of Technology): *Intelligent Agents and Realistic Commercial Application Scenarios*
- Thomas Christaller (Fraunhofer AIS): *Cooperating Teams of Mobile Robots in Dynamic Environments*

A large number of people were involved in making this conference a success. As for the technical program, we thank all those authors who submitted papers to the conference and provided the basis from which we could select a high-quality technical program. The members of the program committee and the additional reviewers helped us in this process by providing timely, qualified reviews and participating in the discussion during the paper selection process. Many thanks to all of you!

We are very grateful to Gerhard Weiss, who served as the Workshop Chair, and Andreas Becks, who served as the Local Arrangements Chair. The Organizing Committee Members took care of the many important details that are needed to make a conference work and that require so much effort and time. Besides Andreas Becks we thank Frank Dylla, Alexander Ferrein, Günter Gans, and Irene Wicke. Günter Gans deserves a special mention for wrestling with the *ConfMan* system and for formatting this volume. We hope you will enjoy its contents!

July 2002 Matthias Jarke, Jana Koehler, and Gerhard Lakemeyer.

KI 2002 Conference Organization

General Chair

Matthias Jarke
RWTH Aachen,
Germany

Program Co-chair

Jana Koehler
IBM Zurich Research Laboratory,
Switzerland

Program Co-chair

Gerhard Lakemeyer
RWTH Aachen,
Germany

Workshop Chair

Gerhard Weiss
TU Munich,
Germany

Organizing Chair

Andreas Becks
Fraunhofer FIT,
Sankt Augustin, Germany

Program Committee

Elisabeth André	Germany	Gerhard Lakemeyer	Germany
Armin Biere	Switzerland	Thomas Malsch	Germany
Susanne Biundo	Germany	Martin Mueller	Canada
Ronen Brafman	Israel	Daniele Nardi	Italy
Gerhard Brewka	Germany	Bernhard Nebel	Germany
Thomas Christaller	Germany	Wolfgang Nejdl	Germany
Sven Dickinson	Canada	Hermann Ney	Germany
Dieter Fox	USA	Juergen Sauer	Germany
Gerhard Friedrich	Austria	Ute Schmid	Germany
Ulrich Furbach	Germany	Dale Schuurmans	Canada
Holger Hoos	Canada	Kilian Stoffel	Switzerland
Ian Horrocks	UK	Michael Thielscher	Germany
Stefan Kirn	Germany	Wolfgang Wahlster	Germany
Jana Koehler	Switzerland	Gerhard Weiss	Germany
Rudolf Kruse	Germany	Stefan Wrobel	Germany

Organizing Committee

Andreas Becks	Fraunhofer FIT, Sankt Augustin	Alexander Ferrein	RWTH Aachen
		Günter Gans	RWTH Aachen
Frank Dylla	RWTH Aachen	Irene Wicke	RWTH Aachen

Additional Referees

Steffen Albrecht
Cyrille Artho
Volker Baier
Peter Baumgartner
Christian Borgelt
Ali Ghodsi Boushehri
Felix Brandt
Ralph Breithaupt
Michael Brenner
Diego Calvanese
Ingo Dahn
Georg Dorffner
Christian Döring
Peter Geibel
Lilia Georgieva
Erich Grädel
Axel Großmann
Jesse Hoey
Tamas Horvath
Geoff Hulten
Luca Iocchi
Dietmar Jannach
Jörg Kindermann
Alexander Kleiner
Aljoscha Klose
Cody Kwok
Edda Leopold
David Lowe
Maren Lübcke
Thomas Lukasiewicz
Silvia Miksch
Bernd Müller
Matthias Nickles
Kai Paetow
Frank Pasemann
Fuchun Peng
Jochen Renz
Jussi Rintanen
Danny Roobaert
Michael Rovatsos
Andrew Rowley
Marco Schmitt
Viktor Schuppan
Klaus Stein
Frieder Stolzenburg
Heiko Timm
Thilo Weigel

KI 2002 Workshops

Workshop Chair

Gerhard Weiss
TU Munich,
Germany

Foundations of Multiagent Systems: The Economic Theory Perspective

Stefan Kirn	Germany
Andreas Will	Germany

Resolving Conflicts between Self-interested Agents (RCSIA 2002)

Ingo J. Timm	Germany
Torsten Eymann	Germany

Modeling Artificial Societies and Hybrid Organizations (MASHO 2002)

Gabriela Lindemann	Germany
Catholijn M. Jonker	The Netherlands
Pietro Panzarasa	U.K.

Workshop on Agent Standards in Theory and Practice (ASTAP 2002)

Michael Berger	Germany
Stefan Kirn	Germany
Jens Nimis	Germany
Ingo J. Timm	Germany
Ubbo Visser	Germany

Workshop on Applications of Description Logics (ADL 2002)

Günther Görz	Germany
Volker Haarslev	Germany
Carsten Lutz	Germany
Ralf Möller	Germany

Workshop on Cognitive Agents

Michael Thielscher	Germany

Table of Contents

Natural Language

LIGHT – A Constraint Language and Compiler System for Typed-Unification Grammars 3
Liviu Ciortuz

Phrase-Based Statistical Machine Translation 18
Richard Zens, Franz Josef Och, Hermann Ney

Compiling Dynamic Agent Conversations 33
Pierre Bonzon

Machine Learning – Combined Approaches

Dynamic Pricing of Information Products Based on Reinforcement Learning: A Yield-Management Approach 51
Michael Schwind, Oliver Wendt

Incremental Fuzzy Decision Trees 67
Marina Guetova, Steffen Hölldobler, Hans-Peter Störr

Learning from Multiple Bayesian Networks for the Revision and Refinement of Expert Systems 82
Michael Borth

Knowledge Representation, Semantic Web, AI Planning

On the Problem of Computing Small Representations of Least Common Subsumers 99
Franz Baader, Anni-Yasmin Turhan

Approximate Information Filtering on the Semantic Web 114
Heiner Stuckenschmidt

ParleE: An Adaptive Plan Based Event Appraisal Model of Emotions 129
The Duy Bui, Dirk Heylen, Mannes Poel, Anton Nijholt

Integrating Function Application in State-Based Planning 144
Ute Schmid, Marina Müller, Fritz Wysotzki

Machine Learning – Neural Networks

Fast Winner-Takes-All Networks for the Maximum Clique Problem 163
Brijnesh J. Jain, Fritz Wysotzki

Augmenting Supervised Neural Classifier Training Using a Corpus of Unlabeled Data 174
Andrew Skabar

Learning of Class Descriptions from Class Discriminations: A Hybrid Approach for Relational Objects 186
Peter Geibel, Kristina Schädler, Fritz Wysotzki

Logic Programming – Theorem Proving – Model Checking

The Well-Founded Semantics Is a Stratified Fitting Semantics 205
Pascal Hitzler, Matthias Wendt

Axiomatization of Finite Algebras 222
Jochen Burghardt

Algorithms for Guiding Clausal Temporal Resolution 235
M. Carmen Fernández Gago, Michael Fisher, Clare Dixon

Vision and Spatial Reasoning

Axiom – A Modular Visual Object Retrieval System 253
Jochen Wickel, Pablo Alvarado, Peter Dörfler, Thomas Krüger, Karl-Friedrich Kraiss

Representation of Behavioral Knowledge for Planning and Plan-Recognition in a Cognitive Vision System 268
Michael Arens, Hans–Hellmut Nagel

Qualitative Velocity and Ball Interception 283
Frieder Stolzenburg, Oliver Obst, Jan Murray

Spatial Inference – Learning vs. Constraint Solving 299
Carsten Gips, Petra Hofstedt, Fritz Wysotzki

Invited Presentations

From Simulated Dialogues to Interactive Performances with Virtual Actors 317
Elisabeth André

Time, Knowledge, and Cooperation: Alternating-Time Temporal Epistemic Logic and Its Applications 318
Michael Wooldridge

Semantic Web Enabled Web Services 319
Dieter Fensel

DFG Priority Programs

DFG Priority Program RoboCup (SPP-1125): Cooperating Teams of Mobile Robots in Dynamic Environments 323
Thomas Christaller

DFG Priority Program (SPP-1083): Intelligent Agents and Realistic Commercial Application Scenarios 324
Stefan Kirn

DFG Priority Program (SPP-1077): Socionics – Investigating and Modelling Artificial Societies 325
Thomas Malsch

Author Index 327

LIGHT – A Constraint Language and Compiler System for Typed-Unification Grammars

L. Ciortuz

CS Department, University of York, UK
ciortuz@cs.york.ac.uk

Abstract. This work presents LIGHT, a feature constraint language for deduction-based bottom-up parsing with typed-unification grammars. We overview both its formal definition, as a logic language operating bottom-up inferences over OSF-terms, and its implementation – an elegant combination of a virtual machine for head-corner parsing and an extended abstract machine for feature structure unification.

1 Introduction

Interest in (typed) unification grammars for Natural Language Processing can be traced back to the seminal work on the PATR-II system [32]. Since then, the *logics* of feature constraints were studied and became well-understood [36, 7], and different types of unification-based *grammar formalisms* were developed by the Computational Linguistics community – most notably Lexical Functional Grammars (LFG, [19]), Head-driven Phrase Structure Grammars (HPSG, [31]) and Categorial (unification) Grammars [39]. More recently *large-scale implementations* for such grammars were devised.

So are for instance the HPSG for English [17] developed at Stanford (called LinGO), two HPSGs for Japanese developed at Tokyo University [24], and respectively at DFKI–Saarbrücken, Germany [34], and the HPSG for German, developed also at DFKI [26]. LFG large-scale grammars were developed by Xerox Corp., but they are not publicly available.

The last years witnessed important advances in the elaboration of techniques for efficient unification and parsing with such grammars [29,30]. The work here presented was part of this international, concerted effort.

The LIGHT language is the only one apart TDL [22] – which proved too much expressive, and therefore less efficient – to be formally proposed for (the systems processing) large-scale LinGO-like grammars. TDL was implemented by the PAGE platform at DFKI – Saarbrücken and the LKB system [16] at CSLI, University of Stanford. They are both interpreters, written in Lisp. LIGHT may be seen as redefining implementing the subset of TDL used in the current versions of LinGO. The LIGHT compiler is one of the (only) two compilers running LinGO, the large-scale HPSG grammar for English. (The other compiler running LinGO is LiLFeS [25]. It basically extends the Prolog unification mechanism to typed feature structures.)

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 3–17, 2002.

There are already a number of papers published about different unification or parsing issues involved in the LIGHT system: [9,13,15,14,12]. However, none of them gave until now a thorough overview of the LIGHT language and system. Besides, we consider that the possibilities for improving the LIGHT system's implementation are far from exhaustion. This is why we found appropriate to publish this overview work about LIGHT.

LIGHT has a fairly elegant logic design in the framework of OSF-logic [4]. The LIGHT name is an acronym for Logic, Inheritance, Grammars, Heads, and Types.[1] Distinctive from the other systems processing LinGO-like grammars, LIGHT uses OSF-theory unification [5] on order- and type-consistent theories. This class of OSF-logic theories extends well-typed systems of feature structures [7], the framework of the other systems processing LinGO. Deduction in LIGHT is the head-corner generalisation [20,35] of bottom-up chart-based parsing-oriented deduction [33].[2]

LIGHT is implemented in C and compiles the input grammar into C code. The implementation is based on a unique combination of an abstract machine for OSF-theory unification – which extends the AM for OSF-term unification [3][9] – and a virtual machine for active bottom-up chart-based parsing [11,14]. The interface between the two machines is structure sharing-oriented. LIGHT uses compiled Quick-Ckeck to speed-up unification [15,23], and a learning module to reduce the size (i.e., the number of constraints) in the rule feature structures [12].

Concerning the organisation of this paper, Sect. 2 presents the logical background of order- and type-consistent OSF-theories, Sect. 3 introduces the formal specifications of the LIGHT language, exemplifying the main notions with excerpts from an HPSG-like typed-unification grammar, and Sect. 4 overviews the implementation of the LIGHT compiler and parsing system.

2 LIGHT Logical Background: Order- and Type-Consistent OSF-Theories

Aït-Kaci, Podelski and Goldstein have introduced in [5] the notion of OSF-theory unification which generalise both OSF-term (ψ-term) unification and well-formed typed feature structure unification [7]. The class of order- and type-consistent OSF-theories introduced in this section extends that of systems of well-formed typed feature structures, henceforth called *well-typed feature structures*. Systems of well-typed feature structures will correspond in fact to order-

[1] The analogy with the name of LIFE – Logic, Inheritance, Functions and Equalities – a well-known constraint logic language based on the OSF constraint system [4] is evident.

[2] For non-lexicalized typed-unification grammars, top-down inferences can be conveniently defined, and in this case LIGHT would be seen as a particular CLP(OSF) language. (For the CLP schema see [18].) It is in fact in this way that LIGHT was first implemented, before it was converted to parsing with large-scale lexicalized unification grammars.

and type-consistent OSF-theories satisfying a set of appropriateness constraints concerning every feature's value and domain of definition.

The advantage of generalising (and subsequently simplifying) the logical framework of typed-unification grammars (from well-typed FSs) to order- and type-consistent OSF-theories is reflected on LIGHT's implementation side by the reduction of the expanded form of LinGO with 42%, and the improving of the average parsing time for the sentences in the *csli* test-suite with 20%.[3]

Let $\mathcal{S}$ be a set of symbols called *sorts*, $\mathcal{F}$ a set of *features*, and $\prec$ a computable partial order relation on $\mathcal{S}$. We assume that $< \mathcal{S}, \prec >$ is a lower semi-lattice, meaning that, for any $s, s' \in \mathcal{S}$ there is a unique greatest lower bound $\text{glb}(s, s')$ in $\mathcal{S}$. This glb is denoted $s \wedge s'$. In the sequel, the notions of sort constraint, feature constraint and equality constraint, OSF-term (or ψ-term, or feature structure/FS, or OSF normalised clause) over the sort signature Σ are like in the OSF constraint logic theory [5]. The same is true for unfolding an OSF-term, and also for subsumption (denoted as $\sqsubseteq$), and unification of two ψ-terms.

Some *notations* to be used in the sequel: $root(\psi)$ and $\psi.f$ denote the sort of the root node in the term ψ, and respectively the value of the feature f at the root level in ψ. The reflexive and transitive closure of the relation $\prec$ will be denoted as $\preceq$.

$Form(\psi, X)$, the *logical form* associated to an OSF-term ψ of the form $s[f_1 \rightarrow \psi_1, ..., f_n \rightarrow \psi_n]$ is $\exists X_1 ... \exists X_n((X.f_1 \doteq Form(\psi_1, X_1) \wedge ... \wedge X.f_n \doteq Form(\psi_n, X_n)) \leftarrow X : s)$, where $\leftarrow$ denotes logical implication, and $X, X_1, ..., X_n$ belong to a countable infinite set $\mathcal{V}$.

An OSF-*theory* is a set of OSF-terms $\{\Psi(s)\}_{s \in \mathcal{S}}$ such that $root(\Psi(s)) = s$, and for any $s, t \in \mathcal{S}$, if $s \not= t$ then $\Psi(s)$ and $\Psi(t)$ have no common variables. The term $\Psi(s)$ will be called the s-sorted type, or simply the s type of the given OSF-theory. A *model* of the theory $\{\Psi(s)\}_{s \in \mathcal{S}}$ is a logical interpretation in which every $Form(\Psi(s), X)$ is valid.

The notion of OSF-term unification is naturally generalised to *OSF-theory unification:* ψ_1 and ψ_2 unify w.r.t. the theory $\{\Psi(s)\}_{s \in \mathcal{S}}$ if there is ψ such that $\psi \sqsubseteq \psi_1, \psi \sqsubseteq \psi_2$, and $\{\Psi(s)\}_{s \in \mathcal{S}}$ entails ψ, i.e., $Form(\psi, X)$ is valid in any model of the given theory.

Example 1. Let us consider a sort signature in which $b \wedge c = d$, and the symbol + is a subsort of *bool*, and the OSF-terms $\psi_1 =$ a[FEAT1 $\rightarrow$ b], $\psi_2 =$ a[FEAT1 $\rightarrow$ c[FEAT2 $\rightarrow$ *bool*]]. The glb of ψ_1 and ψ_2 – i.e., their *OSF-term unification* result – will be $\psi_3 =$ a[FEAT1 $\rightarrow$ d[FEAT2 $\rightarrow$ *bool*]]. Taking $\Psi(d) =$ d[FEAT2 $\rightarrow$ +], the glb of ψ_1 and ψ_2 relative to the $\{\Psi(d)\}$ OSF-theory – i.e., their *OSF-theory unification* result – will be $\psi_4 =$ a[FEAT1 $\rightarrow$ d[FEAT2 $\rightarrow$ +]].

Now we can formalise the link towards well-typed feature structures:

As defined in [5], an OSF-theory $\{\Psi(s)\}_{s \in \mathcal{S}}$ is *order-consistent* if $\Psi(s) \sqsubseteq \Psi(t)$ for any $s \preceq t$. We say that an OSF-theory is *type-consistent* if for any non-atomic

[3] These numbers were (computed after data) obtained and published by U. Callmeier, a former contributor to the LIGHT project [6].

subterm ψ of a $\Psi(t)$, if the root sort of ψ is s, then $\psi \sqsubseteq \Psi(s)$. A term is said to be non-atomic (or: framed) if it contains at least one feature.

A *well-typed* OSF-theory is an order-consistent theory in which the following conditions are satisfied for any $s, t \in \mathcal{S}$:

i. if $f \in$ *Arity(s)* $\wedge f \in$ *Arity(t)*, then
$\exists u \in \mathcal{S}$, such that $s \preceq u, t \preceq u$ and $f \in$ *Arity*(u);
ii. for every subterm ψ in $\Psi(t)$, such that *root*$(\psi) = s$,
if a feature f is defined for ψ, then $f \in$ *Arity*(s), and $\psi.f \sqsubseteq \Psi($*root*$(s.f))$,

where *Arity(s)* is the set of features defined at the root level in the term $\Psi(s)$. An OSF-term ψ satisfying the condition *ii* from above is (said) *well-typed* w.r.t. the OSF theory $\{\Psi(s)\}_{s\in\mathcal{S}}$.

Notes:

1. The condition *i* implies that for every $f \in \mathcal{F}$ there is at most one sort s such that f is defined for s but undefined for all its supersorts. This sort will be denoted *Intro*(f), and will be called the *appropriate domain* on the feature f. Also, *root*$(\Psi(s).f)$, if defined, will be denoted *Approp*(f, s), and will be called the *appropriate value* on the feature f for the sort s. *Approp*$(f,$ *Intro*$(f))$ is the maximal appropriate value for f.[4] The appropriate domain and values for all features $f \in \mathcal{F}$ define the "canonical" *appropriateness constraints* for a well-typed OSF-theory.

2. As a well-typed OSF-theory is (by definition) order-consistent, it implies that *Arity(s)* $\supseteq$ *Arity(t)*, and *Approp(f, s)* $\preceq$ *Approp(f, t)* for every $s \preceq t$;

3. A stronger version for the condition *ii* would be: the feature f is defined (at the root level) for ψ iff $f \in$ *Arity*(s), and $\psi.f \sqsubseteq \Psi(s.f)$. In the latter case, the theory is said to be *totally well-typed*.

For well-typed OSF theories $\{\Psi(s)\}_{s\in\mathcal{S}}$, the notion of OSF-unification extends naturally to *well-typed* OSF-*unification*. The well-typed glb of two feature structures ψ_1 and ψ_2 is the most general (w.r.t. $\sqsubseteq$) well-typed feature structure subsumed by both ψ_1 and ψ_2. The well-typed glb of two feature structures is subsumed by the glb of those feature structures.

To summarise:

The first difference between the class of order- and type-consistent OSF-theories and the class of well-typed OSF-theories concerns the subsumption condition – limited to non-atomic substructures ψ: if *root*$(\psi) = s$, then $\psi \sqsubseteq \Psi(s)$. For instance, if $\mathsf{a[F \rightarrow cons]}$ is type-consistent, its well-typed correspondent is $\mathsf{a[F \rightarrow cons[FIRST \rightarrow top,\ REST \rightarrow list]]}$.

This (more relaxed) condition has been proved to be beneficial for LinGO-like grammars [6], since it lead to a both significant reduction of the expanded

[4] If the lub (least upper bound) of any two sorts exists and is unique, our current implementation of LIGHT uses a weaker version for the condition *i*: if $f \in$ *Arity(s)* $\wedge$ $f \in$ *Arity(t)*, and $\neg\exists u \in \mathcal{S}$ such that $s \preceq u$, $t \preceq u$ and $f \in$ *Arity(u)* then *ApproDom*$(f) = \mathrm{lub}(s, t)$, and *ApproVal*$(f) = \mathrm{lub}($*root*$(s.f),$ *root*$(t.f))$.

Attribute subtyping: $\left.\begin{array}{l} Approp(a,s) = t \\ s' \prec s \end{array}\right\}$ then $Approp(a,s') = t'$, and $t' \preceq t$.

Attribute unique introduction: $\forall f \in \mathcal{F}, \exists^{\star} s \in \mathcal{S}$ and $t.f \uparrow$ for any $t \in \mathcal{S}, s \prec t$.

Type (strict-)consistency: $\left.\begin{array}{l} \psi \text{ is a subtype in } \Psi, \\ root(\psi) = s \end{array}\right\}$ then $\psi \sqsubseteq \Psi(s)$.

Fig. 1. The type discipline in well-typed unification grammars

Order consistency: $s' \prec s$ then $\Psi(s') \sqsubseteq \Psi(s)$.

Type consistency: $\left\{\begin{array}{l} \psi \text{ is a } \textit{non-atomic} \text{ subtype in } \Psi, \\ root(\psi) = s \end{array}\right\}$ then $\psi \sqsubseteq \Psi(s)$.

Fig. 2. The type/FS discipline in order- and typed-consistent grammars

size of the grammar and the parsing time (due to reduction of copying and other structure manipulation operations), without needing a stronger notion of unification.

The second main difference between order- and type-consistent OSF-theories on one side, and well-typed OSF-theories on the other side is related to appropriate features: well-typed theories do not allow a subterm ψ of root sort s to use features not defined at the root level in the corresponding type $\Psi(s)$.

For instance, if ψ_5 = a[FEAT1 $\rightarrow$ d[FEAT2 $\rightarrow$ +, FEAT3 $\rightarrow$ *bool*]], then the OSF-theory glb of ψ_2 and ψ_5 will be defined (and equal to ψ_5), while their well-typed glb relative to the same theory does not exist, simply because ψ_5 is not well-typed w.r.t. $\Psi(d)$, due to the non-appropriate feature FEAT3.

Therefore, LIGHT will allow the grammar writer more freedom. The source of this freedom resides in the *openness* of OSF-terms. Also, at the implementation level, LIGHT works with free-order registration of features inside feature frames.[5]

Just to get a comparative overview on 1. well-typed unification grammars [7] versus 2. order-and type-consistent OSF-theories/grammars, we synthesise the definitions for 1. the appropriateness constraints in Fig. 1, and respectively 2. the order- and type-consistency notions in Fig. 2. It is obvious that our approach is simpler than introduced in [7].

A procedure – called *expansion* – for automate transformation of an OSF-theory into an order- and type-consistent one was presented in [13]. This proce-

[5] The AMALIA and LiLFeS abstract machines for unification of well-typed FS work with closed records and fixed feature order for a given type. Recent work by Callmeier has -shown that fix-order feature storing does not lead to improvement of the parse performance on LinGO.

dure also computes the "canonical" set of appropriate constraints associated to the expanded theory. OSF-theory unification for an order- and type-consistent theory is well-formed with respect to this canonical set of appropriate constraints. With respect to the canonical appropriateness constraints induced for an order- and type-consistent OSF-theory (in particular, for LinGO), well-typed feature structure unification coincides with OSF-theory unification.

3 Formal Introduction to LIGHT

3.1 Constraint Logic Definitions

A LIGHT logic grammar will be defined as a particular order- and type-consistent OSF-theory. (For an introduction to logic grammars see [1].)

Let $< \mathcal{S}, \mathcal{F}, \prec >$ be a sort signature, with $\mathcal{S}$ containing several "reserved" sorts, namely *top, sign, rule-sign, lexical-sign* and *start*, and $\mathcal{F}$ the "reserved" features STEM and ARGS. The sorts *rule-sign* and *lexical-sign* – both being descendants of *sign* – are disjunct, while *start* is a descendant of *rule-sign*. $\mathcal{S}$ is assumed a lower semi-lattice w.r.t. $\prec$.

- A LIGHT Horn clause, denoted $\psi_0 :- \psi_1\ \psi_2\ ...\ \psi_n$ $(n \geq 0)$, corresponds to the logical formula $\tilde{\forall}(\psi_0 \leftarrow \psi_1 \wedge \psi_2 \wedge ... \wedge \psi_n)$, where $\psi_i, i = 0, 1, 2, ..., n$ are ψ-terms, $root(\psi_0) \preceq$ *rule-sign*, and $root(\psi_i) \preceq$ *rule-sign* or $root(\psi_i) \preceq$ *lexical-sign* for $i = 1, 2, ..., n$. (Remark the absence of predicate symbols.)
- A LIGHT *rule* is a LIGHT Horn clause with $root(\psi_0)$ a leaf node in the sort hierarchy $(\mathcal{S}, \prec)$.[6] (The STEM feature in rules' definition is related to the consumed input sentence.)

Remark: Any LIGHT rule can be written as a single ψ-term, if we denote the right hand side (RHS) arguments as a list value of the (reserved) feature ARGS. Anywhere we refer to a rule as an OSF-term, we assume this understanding.

- A LIGHT *logic grammar* is an order- and type-consistent OSF-theory containing a non-empty set of LIGHT rules. (In practice we require in fact that all leaf *rule-sign*-descendants types be LIGHT rules.)

3.2 Parsing-Oriented Derivation Specific Definitions

For bottom-up chart-based parsing, [33] and [35] propose two derivation rules: *scan* and *complete*. Head-corner parsing [35] distinguishes between *left-* and *right-*scan and respectively complete, and adds a new derivation rule, *head-corner*. At one rule's application, unification with the head-corner argument, designated by the grammar writer, is tried before the other arguments are treated. This "head" argument is the most important one, because it is usually critical for the

[6] For the moment, the LIGHT language is not designed to deal with so-called ϵ-rules, i.e., rules whose right hand side (RHS) is empty.

application of the whole rule, i.e., statistically speaking, it is most probable to produce unification failure. It is worth to note that in a (order-)sorted framework, the distinction between terminal and non-terminal symbols is erased, since either a terminal or a non-terminal may occupy a same slot in the ARGS list of a rule. In conclusion, in the LIGHT setup, head-corner bottom-up chart-based parsing is achieved via two deduction rules: head-corner and (left- and right-) complete.[7]

Lexical Analysis:

Let $< w_1 w_2 \ ... \ w_n >$ be an input (word) sequence. If ψ is a leaf *lexical-sign*-descendant in the grammar $\mathcal{G}$ such that $\psi.\text{STEM} = < w_i w_{i+1} \ ... \ w_j >$, with $1 \leq i \leq j \leq n$, then the quadruple $(\epsilon, \psi', i-1, j)$, where ψ' is obtained from ψ by renaming all its variables with new variables, is a *lexical item*. Here, ϵ is the empty set (of atomic OSF-constraints). Any lexical item is a *passive* item.

In LinGO, the lexical analysis is slightly more elaborated: a *lexical-rule*, which is a leaf descendant of the *lexical-rule* sort (disjunct from *rule-sign*) may be applied during the morphological analysis to a leaf *lexical-sign*-descendant. The resulting feature structure takes the role of the above ψ'.

Syntactic Analysis:

Head-Corner: If (σ, ψ, i, j) is a passive item, $\psi_0 :- \ \psi_1 \ ... \ \psi_r$ is a newly renamed instance of a rule in $\mathcal{G}$, and ψ_k is the head/key argument of this rule $(1 \leq k \leq r)$, then if there is a glb φ of ψ_k and ψ, $(1 \leq k \leq r)$ with τ the subsumption substitution of ψ_k into $\varphi = \text{glb}(\psi_k, \psi)$, i.e., $\tau\psi_k = \text{glb}(\psi_k, \psi)$, then $(\tau\sigma, \psi_0 :- \ \psi_1 \ ... \ .\psi_k. \ ... \ \psi_r, i, j)$ is an item. It is an *active* item if $r > 1$, and a passive item if $r = 1$. The compositions $\tau\sigma$ and $\sigma\psi$ must be interpreted simply as a conjunction/union of the formulae/sets of atomic OSF-constraints τ and σ, respectively σ and ψ.

Right Complete: If $(\sigma, \psi_0 :- \ \psi_1 \ ... \ .\psi_p \ ... \ \psi_q. \ ... \ \psi_r, i, j)$ is an active item, with $q < r$, and there is a passive item, either a lexical (τ, ψ, j, k) or a non-lexical one $(\tau, \psi :- \ .\psi'_1 \ ... \ \psi'_m., j, k)$, assuming that $\text{glb}(\tau\psi, \sigma\psi_{q+1})$ exists, and υ is the corresponding subsumption substitution, then $(\upsilon\sigma, \psi_0 :- \ \psi_1 \ ... \ .\psi_p \ ... \ \psi_{q+1}. \ ... \ \psi_r, i, k)$ is an item, namely a passive one if $p = 1$ and $q + 1 = r$, respectively an active one, otherwise.

Left Complete: Similar to the above definition, except that the active item is leftward "open". If $(\sigma, \psi_0 :- \ \psi_1 \ ... \ .\psi_p \ ... \ \psi_q. \ ... \ \psi_r, i, j)$ is an active item, with $1 < p$, and there is a passive item, either a lexical (τ, ψ, k, i) or a non-lexical one $(\tau, \psi :- \ .\psi'_1 \ ... \ \psi'_m., k, i)$, assuming that $\text{glb}(\tau\psi, \sigma\psi_{q+1})$ exists, and υ is the corresponding subsumption substitution, then $(\upsilon\sigma, \psi_0 :- \ \psi_1 \ ... \ .\psi_{p-1} \ ... \ \psi_q. \ ... \ \psi_r, k, j)$ is an item, namely a passive one if $p - 1 = 1$ and $q = r$, respectively an active one, otherwise.

A LIGHT *parse sequence* (or *derivation*) is a finite sequence of items $i_1, ..., i_m$ such that for every r with $1 \leq r \leq m$, the item i_r is either a lexical item, or a head-corner production based on a passive item i_q with $q < r$, or it is obtained

[7] LinGO uses only binary rules, therefore it became convenient to use the name *complete* as a "rewriting" one for both the classical scan and complete parsing operations, when extended to order-sorted unification based-parsing.

```
satisfy_HPSG_principles
[ STEM   diff_list,
  CAT    #1:categ,
  SUBCAT #2:categ_list,
  HEAD   #4:phrase_or_word
            [ STEM   diff_list,
              CAT #1,
              SUBCAT #3|#2 ],
  COMP   #5:phrase_or_word
            [ STEM   diff_list,
              CAT #3,
              SUBCAT nil ],
  ARGS   <#4, #5> ]
```

```
girl
[ STEM   <!"girl"!>,
  CAT    noun,
  SUBCAT <det> ].
```

Fig. 3. A sample rule (parent) type (satisfy_HPSG_principles) and a lexical entry (girl)

by left- or right- completion of an active item i_p with a passive item i_q, where $p, q < r$.

Let $w = < w_1, w_2, ..., w_n >$ be a finite sequence of symbols (from $\mathcal{S}$). The LIGHT logic grammar $\mathcal{G}$ *recognises* the input sentence w iff there is a parse sequence (derivation) in $\mathcal{G}$ ending with a passive item $(\sigma, \psi, 0, n)$, where ψ is *start*-sorted.

3.3 Exemplification:

The satisfy_HPSG_principles feature structure adapted from [37] and presented in Fig. 3 encodes the Head Feature Principle (CAT $\doteq$ HEAD.CAT), The Saturation Principle (COMP.SUBCAT:nil), and the Subcategorization Principle (to be later detailed). These three principles are among the basic ones in the HPSG linguistic theory [31]. The syntax used in Fig. 3 is that of LIGHT:

The sorts start, list and the list subsorts cons and nil are special (reserved LIGHT) types, and so is diff_list, the difference list sort. The notation <! !> is a syntax sugar for difference lists, just as < > is used for lists.[8] The symbol | is used as a constructor for non-nil (i.e., cons) lists.

The linguistic significance of the Head Feature Principle: the CATegory of a phrase is that of (or: given by) its head. For the Saturation Principle: the complement of a phrase must be saturated – procedurally: it must have been fully parsed – at the moment of its integration into a (larger) phrase. The Subcategorization Principle correlates the SUBCAT feature value – namely a list of categories – for the HEAD argument with the SUBCAT feature value of the phrase itself, and the CAT feature value for the complement (COMP) daughter of the same phrase.

[8] Formally, $<!a_1, a_2, ..., a_n!>$ stands for

$$\text{diff\_list}[\ \text{FIRST\_LIST}\ a_1|a_2|...|a_n|\#1,\ \text{REST\_LIST}\ \#1\ ]$$

The Subcategorization Principle is directly linked to the nature of the parsing itself in such lexicalized typed-unification grammars: no specific parsing rules are provided; only very general principles/constraints are given about how phrases or words may be combined, while specific information about the combinatorial valances of words is provided in the lexical descriptions.

Basically, the Subcategorization Principle says that, when applying a rule – in order to build a *mother* feature structure out of a *head daughter* and a *complement daughter* –, the complement daughter "consumes" the first element of the head daughter's SUBCAT list, and "transmits" the rest of that list to the mother feature structure.

The satisfy_HPSG_principles type will transmit via inheritance the constraints it incorporates to the rules used in the [37] grammar, namely two binary rules: lh_phrase and rh_phrase. The constraints specific to these rules will impose only that the head is on the left, respectively the right position inside a phrase.

4 LIGHT – Implementation

4.1 Related Systems

Systems processing typed-unification grammars opted for different approaches. ALE [8] opted for the translation of typed FSs into Prolog terms. $\mathcal{A}$MALIA [40, 41] offered the first view on compiling ALE grammars – in which typed FSs had a less general form than in the LinGO-like HPSG grammars –, based on an abstract machine which adapts WAM [2] to FS unification and replaces SLD-resolution with simple bottom-up chart-based parsing. LiLFeS [25], the other compiler (besides LIGHT) running LinGO followed in the beginning the same direction, but then processed the grammar rules so to apply well-known CFG parsing algorithms. (Later developments for TDL also explored CFG approximations of LinGO-like grammars [21].) The LKB[16], TDL [22] systems' basic versions, and PET [6] implemented simple head-corner parsers, opting for different versions of FS unifiers [38,42,23]. Later, hyper-active parsing [27] and ambiguity packing [28] were incorporated into LKB and PET.

For LIGHT, we chose to extend the AM for unification of OSF-terms [3] to OSF-theory unification [5], making it implicitly able to unify typed FSs [7]. For parsing, while originally the LKB simple head-corner active bottom-up chart-based parser was imported in our system, later on we replaced it with a VM for incremental head-corner bottom-up parsing with FS sharing and backtracking [11]. The interesting point about this VM is that on one hand it can be used by the LKB-like systems (if they would like to opt for FS sharing), and on the other hand it can work as an interpreter of the abstract code (for rules) generated by the LIGHT compiler.

4.2 LIGHT System's Architecture

An overview of our LIGHT parser's architecture is shown in Fig. 4. The LIGHT compiler translates typed-unification grammars (written or converted into a format inspired by the OSF notation) into abstract code, and then into C. Each

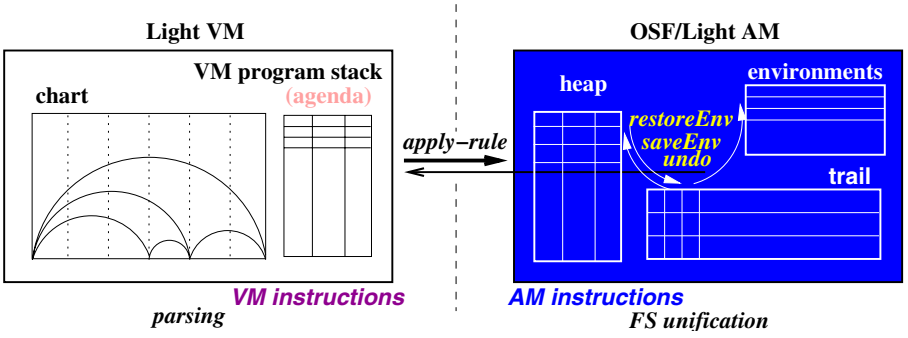


Fig. 4. An overview of the VM for HC bottom-up chart-based parsing

VM Instructions		*AM Instructions*	
parsing	*interface*	*READ*-stream	*WRITE*-stream
keyCorner	undo	push_cell	intersect_sort
directComplete	saveEnvironment	set_sort	test_feature
reverseComplete	restoreEnvironment	set_feature	unify_feature
apply_rule		write_test	

Fig. 5. Instructions in LIGHT VM and OSF/LIGHT AM

typed FS in the input grammar gets an associated (compiled) C function. The FSs representing rules undergo a further, automate transformation of the abstract code so to make them suitable for efficient head-corner bottom-up chart-based parsing with FS sharing. We refer to this transformation scheme as Specialised Rule Compilation (SRC) of rules [14].[9]

Each of the basic operations for parsing in LIGHT – a rule application, a lexical FS construction, or a type-checking (we call it: on-line expansion) – are achieved by calling a compiled function. The abstract machine instructions which build up these functions are shown in the right side of the table in Fig. 5.

As the AM in [3] was concerned only with OSF-term unification – this AM will be referred to in the sequel as OSF AM –, we extended the definitions of the two of its AM instructions, namely intersect_sort and test_feature, and also the *bind-refine* procedure invoked by the on-line unifier function *osf-unify*

[9] The execution of the abstract code for rules in LIGHT has an effect similar to that produced by AMALIA on ALE grammars, namely it achieves bottom-up chart-based parsing. Unlike AMALIA, we produce this ("specialised") code through automate transformation of the (non "specialised") abstract code produced for the FS representing the rule [14]. (Note that rules in LinGO are represented as FSs; their arguments constitute the value of the ARGS reserved feature/attribute.) Additionally, the SRC-optimised code in LIGHT incorporates specialised sequences for dealing with environments, for feature structure sharing.

in [3], in order to support on-line expansion/type-checking needed for OSF-theory unification. Besides these transformations, we incorporated into OSF AM some other facilities: tracing, backtracking, and FS sharing. We use the name OSF/LIGHT AM to denote the new, more general version of OSF AM. For full details our work extending OSF AM to OSF/LIGHT AM, the interested reader must refer to [9]. OSF AM and consequently OSF/LIGHT AM imported the two-stream (*READ* and *WRITE*) optimisation from the Warren Abstract Machine [2]. Thus the set_sort and intersect_sort abstract instructions correspond to sort constraints, set_feature and test_feature correspond to feature constraints, and unify_feature corresponds to equation constraints.

The main data structures in the OSF/LIGHT AM are a heap for building up FS representation, a trail which registers the modifications on the heap during unification-achieving operations, and an array of environments used for FS sharing.

The parsing (i.e., the control above rules' application) is implemented in the LIGHT system as a virtual machine hereby referred to as LIGHT VM. The main data structures in LIGHT VM are a chart and the VM's program stack, which replaces in a quite elaborated way the well-known *agenda* in chart-based parsing.

Other data structures in LIGHT VM:

– the rule (syntactic and lexical) *filters*,
– the *dictionary*, which associates to every word which is a *root* in at least one lexical entry the list of all lexical entries for which that word is the root,
– the *lexicon*, which associates every lexical entries the index of a *query* (compiled) function and eventually the index of a lexical rule to be applied to the FS constructed by that query function,
– the pre-compiled *QC-arrays* associated to rules [15].

The LIGHT VM's instructions are listed in the first column of the table in Fig. 5. (The interested reader will find their detailed description in [11].)

The apply_rule function appearing on the bottom of the first column in Fig. 5 is not a VM instruction (this is why we delimited it from above by a horizontal line). It is called by each of the three VM parsing instructions – keyCorner, directComplete, and reverseComplete. Conceptually, it acts in fact like a higher level function, which applies the compiled function corresponding to a (specified) rule FS to a certain argument.

Instead, apply_rule is part of the *interface* between the LIGHT VM and the OSF/LIGHT AM. This interface contains also three procedures implemented within the unifier (in our case: OSF/LIGHT AM), which will be applied/called from within the VM program: undo, saveEnvironment and restoreEnvironment. The undo procedure performs backtracking to a certain state of the AM (namely as it was before an unification attempt), restoring the information corresponding to a successful unification (according to FS sharing scheme). The saveEnvironment procedure performs the same task as undo, but also moves a certain amount of data from the trail to a new environment, while restoreEnvironment performs the opposite operation.

Overall:	memory consumption	process size full/resident	average parsing time
regular compilation	59.5MB	73MB/80MB	128 msec
specialised compilation	3.9MB	44MB/13MB	35 msec

Detailed:	heap cells	feature frames	environments	trail cells	corefe-rences
regular compilation	1,915,608	1,050,777	2669	128,747	0
specialised compilation	77,060	57,663	2669	77,454	22,523
GR-optimisation	37,915	29,908	2669	44,424	12,137

Fig. 6. Regular vs. specialised rule compilation: a comparison between the respective effects on parsing the *csli* test-suite. (Further memory reduction due to GR is added)

4.3 Final Notes on LIGHT Implementation

Both the OSF/LIGHT AM for unification and the LIGHT VM for head-corner parsing can be used separately, namely just for unifying FSs, or respectively to achieve parsing using another unifier (than the OSF/LIGHT AM). Two parsers co-exist in the LIGHT system, one corresponding to the Specialised Rule Compilation (SRC) optimisation, the other using as unification means only the *osf-unify* procedure [3] upgraded with type-checking/on-line expansion.[10] The two parsers are implemented in (or as instances of) the same VM, just by changing the definition of the higher-level function rule.

The speed-up factor provided by implementing the SRC optimisation is 3.66 when running the LinGO grammar on the *csli* test-suite. A subsequent 43% speed-up was provided the compilation of the QC pre-unification filter [15], while recently, the Generalised Reduction (GR) learning technique working in connection with two-step unification further sped-up parsing up to 23% [12]. LIGHT's current best parsing time was an average of 18.4 msec. per sentence on the *csli* test-suite, registered on a Pentium III PC at 933MHz running Red Hat Linux 7.1. The compilation of the LinGO grammar (the *CLE* version) on that computer takes 79 seconds, including the filter computation (which is the most consuming task in grammar processing).[11]

Our SRC-optimisation also reduced dramatically the memory space used during parsing, as one can see in Fig. 6.

[10] Therefore the second parser does not appeal the real power of the compiler; it serves for basic tests, and – most important – as a support for developing new extensions and improved versions of the LIGHT compiler.

[11] By the time LIGHT system's development was frozen due to the author's departure from DFKI – Saabrbrücken (at the end of the year 2000), the LIGHT parsing speed has been proved slightly higher than that of the fastest (and since then, commercially supported) interpreter for LinGO-like grammars – namely the PET system [6].

5 Conclusion and Further Work

This paper presented LIGHT, a feature constraint language in the framework of OSF-logic, which underlines large-scale typed-unification grammars. Until now, LIGHT was used for parsing with LinGO [17], the HPSG grammar for English developed at CSLI, University of Stanford, and for automate learning of typed-unification grammars [10].

The implementation of LIGHT is based on an interesting combination of a virtual machine for head-corner bottom-up chart-based parsing and an abstract machine for (typed) FS unification. Other differences with respect to the other systems implementing LinGO-like typed-unification grammars are environment-based FS sharing, incremental parsing and compiled Quick-Check [23].

We expect that further major improvements to the LIGHT compiler will come following the finding (on the LinGO grammar) that there are usually a reduced number of paths (the so-called *GR-paths*) that contribute to unification failure [12]. This fact may be further exploited so to *i.* fully integrate QC into compiled unification; *ii.* eliminate the need for the – much time-consuming – FS restoring operation, currently part of the FS sharing mechanism; and *iii.* replace searching through feature frames with direct access into vectors of GR-path values associated to each rule.

Independent of the developments suggested above, further fine engineering the LIGHT system – for instance making the computations be done as locally as possible – is supposed to significantly improve the current performances. Finally, we expect that the system can be adapted to run other kind of applications, like deductive frame-oriented databases and ontologies.

Acknowledgements. The LIGHT system was developed at the Language Technology Laboratory of The German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany. U. Callmeier contributed to the implementation of the head-corner bottom-up (non VM-based) parser for CHIC, which was the development prototype of LIGHT. The present paper was written while the author was supported by an EPSRC ROPA grant at the University of York.

References

1. H. Abramson and V. Dahl. *Logic Grammars*. Symbolic Computation AI Series. Springer-Verlag, 1989.
2. H. Aït-Kaci. *Warren's Abstract Machine: A Tutorial Reconstruction*. The MIT Press, Cambridge, MA, 1991.
3. H. Aït-Kaci and R. Di Cosmo. Compiling order-sorted feature term unification. Technical report, Digital Paris Research Laboratory, 1993. PRL Technical Note 7, downloadable from `http://www.isg.sfu.ca/life/`.
4. H. Aït-Kaci and A. Podelski. Towards a meaning of LIFE. *Journal of Logic Programming*, 16:195–234, 1993.
5. H. Aït-Kaci, A. Podelski, and S.C. Goldstein. Order-sorted feature theory unification. *Journal of Logic, Language and Information*, 30:99–124, 1997.

6. U. Callmeier. PET – a platform for experimentation with efficient HPSG processing techniques. *Journal of Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):99–108, 2000.
7. B. Carpenter. *The Logic of Typed Feature Structures – with applications to unification grammars, logic programs and constraint resolution.* Cambridge University Press, 1992.
8. B. Carpenter and G. Penn. ALE: The Attribute Logic Engine. User's Guide. Technical report, Carnegie-Mellon University. Philosophy Department. Laboratory for Computational Linguistics, Pittsburgh, 1992.
9. L. Ciortuz. LIGHT – another abstract machine for FS unification. In D. Flickinger, S. Oepen, J. Tsujii, and H. Uszkoreit, editors, *Collaborative Language Engineering.* CSLI Publications, The Center for studies of Language, Logic and Information, Stanford University, 2002.
10. L. Ciortuz. Towards ILP-based learning of attribute path values in typed-unification grammars. 2002. (Submitted).
11. L. Ciortuz. A virtual machine design for head-corner parsing with feature structure sharing. 2002. (Submitted).
12. L. Ciortuz. Learning attribute values in typed-unification grammars: On generalised rule reduction. In *Proceedings of the 6th Conference on Natural Language Learning (CoNLL–2002)*, Taipei, Taiwan, 31 August – 1 September 2002. Morgan Kaufmann Publishers and ACL.
13. L. Ciortuz. Expanding feature-based constraint grammars: Experience on a large-scale HPSG grammar for English. In *Proceedings of the IJCAI 2001 co-located Workshop on Modelling and solving problems with constraints*, Seattle, USA, August 4–6, 2001. Downloadable from `http://www.lirmm.fr/~bessiere/proc_wsijcai01.html`.
14. L. Ciortuz. On compilation of head-corner bottom-up chart-based parsing with unification grammars. In *Proceedings of the IWPT 2001 International Workshop on Parsing Technologies*, pages 209–212, Beijing, China, October 17–19, 2001.
15. L. Ciortuz. On compilation of the Quick-Check filter for feature structure unification. In *Proceedings of the IWPT 2001 International Workshop on Parsing Technologies*, pages 90–100, Beijing, China, October 17–19, 2001.
16. A. Copestake. *The (new) LKB system.* CSLI, Stanford University, 1999.
17. A. Copestake, D. Flickinger, and I. Sag. *A Grammar of English in HPSG: Design and Implementations.* Stanford: CSLI Publications, 1999.
18. J. Jaffar and M. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19(20):503–582, May-July 1994.
19. R. M. Kaplan and J. Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–381. The MIT Press, 1982.
20. M. Kay. Head driven parsing. In *Proceedings of the 1st Workshop on Parsing Technologies*, pages 52–62, Pittsburg, 1989.
21. B. Kiefer and H-U. Krieger. A context-free approximation of Head-driven Phrase Structure Grammar. In *Proceedings of the 6th International Workshop on Parsing Technologies*, pages 135–146, Trento, Italy, 2000.
22. H.-U. Krieger and U. Schäfer. TDL – A Type Description Language for HPSG. Research Report RR-94-37, German Research Center for Artificial Intelligence (DFKI), 1994.
23. R. Malouf, J. Carroll, and A. Copestake. Efficient feature structure operations without compilation. *Journal of Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):29–46, 2000.

24. Y. Mitsuishi, K. Torisawa, and J. Tsujii. HPSG-Style Underspecified Japanese Grammar with Wide Coverage. In *Proceedings of the 17th International Conference on Computational Linguistics: COLING-98*, pages 867–880, 1998.
25. Y. Miyao, T. Makino, K. Torisawa, and J. Tsujii. The LiLFeS abstract machine and its evaluation with the LinGO grammar. *Journal of Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):47–61, 2000.
26. Stefan Müller. *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche.* Number 394 in Linguistische Arbeiten. Max Niemeyer Verlag, Tübingen, 1999.
27. S. Oepen and J. Caroll. Performance profiling for parser engineering. *Journal of Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG: Methods, Systems, Evaluation):81–97, 2000.
28. S. Oepen and J. Carroll. Ambiguity packing in HPSG – practical results. In *Proceedings of the 1st Conference of the North American Chapter of the ACL*, pages 162–169, Seattle, WA, 2000.
29. S. Oepen, D. Flickinger, H. Uszkoreit, and J. Tsujii, editors. *Special Issue on Efficient Processing with HPSG: Methods, Systems, Evaluation.* Cambridge University Press, 2000. Journal of Natural Language Engineering, 6 (1).
30. S. Oepen, D. Flickinger, H. Uszkoreit, and J. Tsujii, editors. *Collaborative Language Engineering.* CSLI Publications, University of Stanford, CA, 2002.
31. C. Pollard and I. Sag. *Head-driven Phrase Structure Grammar.* Center for the Study of Language and Information, Stanford, 1994.
32. S. M. Shieber, H. Uszkoreit, F. C. Pereira, J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In J. Bresnan, editor, *Research on Interactive Acquisition and Use of Knowledge.* SRI International, Menlo Park, Calif., 1983.
33. S.M. Shieber, Y. Schabes, and F. Pereira. Principles and implementation of deductive parsing. *Jornal of Logic Programming*, pages 3–36, 1995.
34. M. Siegel. HPSG analysis of Japanese. In *Verbmobil: Foundations of Speech-to-Speech Translation.* Springer Verlag, 2000.
35. N. Sikkel. *Parsing Schemata.* Springer Verlag, 1997.
36. G. Smolka. Feature-constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87, 1992.
37. G. Smolka and R. Treinen, editors. *The DFKI Oz documentation series.* German Research Center for Artificail Intelligence (DFKI), Stuhlsatzenhausweg 3, Sarrbrücken, Germany, 1996.
38. H. Tomabechi. Quasi-destructive graph unification with structure-sharing. In *Proceedings of COLING-92*, pages 440–446, Nantes, France, 1992.
39. H. Uszkoreit. Categorial Unification Grammar. In *International Conference on Computational Linguistics (COLING'92)*, pages 498–504, Nancy, France, 1986.
40. S. Wintner. *An Abstract Machine for Unification Grammars.* PhD thesis, Technion – Israel Institute of Technology, 32000 Haifa, Israel, 1997.
41. S. Wintner and N. Francez. Efficient implementation of unification-based grammars. *Journal of Language and Computation*, 1(1):53–92, 1999.
42. D. A. Wroblewski. Non-destructive graph unification. In Dalle Miller, editor, *Proceedings of the 6th national conference on artificial intelligence (AAI'87)*, pages 582–587, Seattle, 1987.

Phrase-Based Statistical Machine Translation

Richard Zens, Franz Josef Och, and Hermann Ney

Human Language Technology and Pattern Recognition
Lehrstuhl für Informatik VI
Computer Science Department
RWTH Aachen – University of Technology
Germany

Abstract. This paper is based on the work carried out in the framework of the VERBMOBIL project, which is a limited-domain speech translation task (German-English). In the final evaluation, the statistical approach was found to perform best among five competing approaches.
In this paper, we will further investigate the used statistical translation models. A shortcoming of the single-word based model is that it does not take contextual information into account for the translation decisions. We will present a translation model that is based on bilingual phrases to explicitly model the local context. We will show that this model performs better than the single-word based model. We will compare monotone and non-monotone search for this model and we will investigate the benefit of using the sum criterion instead of the maximum approximation.

1 Introduction

In this paper, we will study some aspects of the phrase-based translation (PBT) approach in statistical machine translation. The baseline system we are using has been developed in the VERBMOBIL project [17].

In the final project evaluation [13], several approaches were evaluated on the same test data. In addition to a classical rule-based approach [4] and a dialogue-act based approach [12] there were three data-driven approaches, namely an example-based [1], a substring-based [2] and a statistical approach developed in the authors' group. The data-driven approaches were found to perform significantly better than the other two approaches. Out of the data-driven approaches the statistical approach performed best, e.g. the error rate for the statistical approach was 29% instead of 62% for the classical rule-based approach.

During the progress of the VERBMOBIL project, different variants of statistical translation systems have been implemented and experimental tests have been performed for text and speech input [7,10]. The major variants were:

- the single-word based approach (SWB), see Sect. 2.2
- the alignment template approach (AT), see Sect. 2.3

The evaluation showed that the AT approach performs much better than the SWB variant. It is still an open question, which components of the AT system are responsible for the improvement of the translation quality.

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 18–32, 2002.

In this paper, we will review the baseline system and we will describe in detail a method to learn phrasal translations. We will compare SWB to phrase-based translation, monotone to non-monotone search, and the sum criterion to maximum approximation.

2 Review of the Baseline System

2.1 Bayes Decision Rule

The goal of machine translation is to automatically transfer the meaning of a source language sentence $f_1^J = f_1, ..., f_j, ..., f_J$ into a target language sentence $e_1^I = e_1, ..., e_i, ..., e_I$. In statistical machine translation, the conditional probability $Pr(e_1^I|f_1^J)$[1] is used to describe the correspondence between the two sentences. This model can be used directly for translation by solving the following maximization problem:

$$\hat{e}_1^I = \underset{e_1^I}{\operatorname{argmax}} \left\{ Pr(e_1^I|f_1^J) \right\} \tag{1}$$

$$= \underset{e_1^I}{\operatorname{argmax}} \left\{ Pr(e_1^I) \cdot Pr(f_1^J|e_1^I) \right\} \tag{2}$$

In the second equation, we have applied Bayes theorem. The decomposition into two knowledge sources makes the modeling easier. Now, we have two models:

1. the language model $Pr(e_1^I)$ and
2. the translation model $Pr(f_1^J|e_1^I)$.

The language model describes the correctness of the target language sentence. It helps to avoid syntactically incorrect sentences. A detailed description of language models can be found in [6]. This paper will focus on the translation model.

The resulting architecture for the statistical translation approach is shown in Fig. 1 with the translation model further decomposed into lexicon and alignment model.

2.2 Single Word-Based Translation Models

Concept. A key issue in modeling the string translation probability $Pr(f_1^J|e_1^I)$ is the question of how we define the correspondence between the words of the target sentence and the words of the source sentence. In typical cases, we can assume a sort of pairwise dependence by considering all word pairs (f_j, e_i) for a given sentence pair $(f_1^J; e_1^I)$. Models describing these types of dependencies are referred to as alignment models [3,16].

When aligning the words in parallel texts, we typically observe a strong localization effect. Figure 2 illustrates this effect for the language pair German-English. In many cases, although not always, there is an additional property: over large portions of the source string, the alignment is monotone.

[1] The notational convention will be as follows: we use the symbol $Pr(\cdot)$ to denote general probability distributions with (nearly) no specific assumptions. In contrast, for model-based probability distributions, we use the generic symbol $p(\cdot)$.

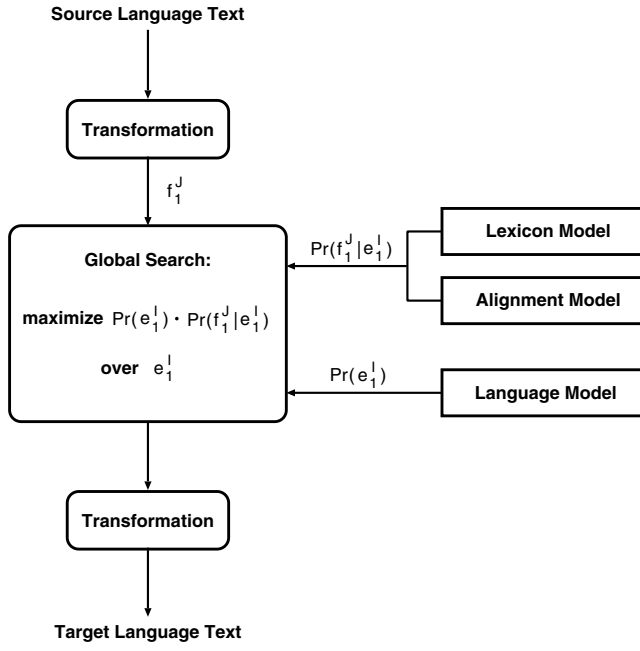


Fig. 1. Architecture of the translation approach based on Bayes decision rule

Basic Alignment Models. To arrive at a quantative specification, we define the alignment mapping: $j \rightarrow i = a_j$, which assigns a word f_j in position j to a word e_i in position $i = a_j$. We rewrite the probability for the translation model by introducing the 'hidden' alignments $a_1^J := a_1...a_j...a_J$ for each sentence pair $(f_1^J; e_1^I)$. to structure this probability distribution, we factorize it over the positions in the source sentence and limit the alignment dependencies to a first-order dependence:

$$Pr(f_1^J|e_1^I) = \sum_{a_1^J} Pr(f_1^J, a_1^J|e_1^I) \tag{3}$$

$$= \sum_{a_1^J} Pr(a_1^J|e_1^I) \cdot Pr(f_1^J|a_1^J, e_1^I) \tag{4}$$

$$= p(J|e_1^I) \cdot \sum_{a_1^J} \prod_{j=1}^{J} [p(a_j|a_{j-1}, I, J) \cdot p(f_j|e_{a_j})] \tag{5}$$

Here, we have the following probability distributions:

- the sentence length probability: $p(J|e_1^I)$, which is included here for completeness, but can be omitted without loss of performance;
- the lexicon probability : $p(f|e)$;
- the alignment probability: $p(a_j|a_{j-1}, I, J)$.

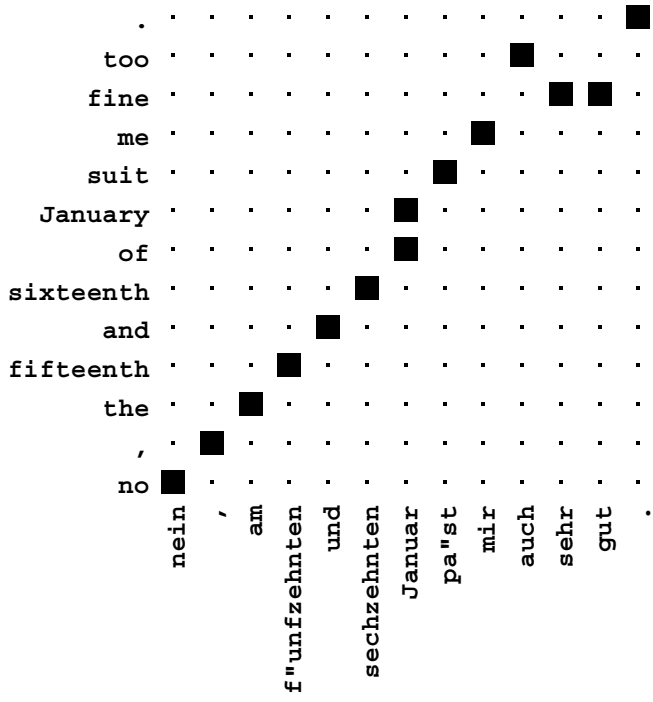


Fig. 2. Word aligned sentence pair

There are various ways to model the alignment probability. For further details, see [3,16]. Some of these models take one-to-many alignments explicitly into account, but the lexicon probabilities $p(f|e)$ are still based on single words in each of the two languages.

We will show the results for two search variants: a monotone search (later referred to as MON) that produces only monotone translations and a quasi-monotone search procedure (later referred to as GE). This proceeds from left to right along the positions of the source sentence but allows for a small number of source positions that are not processed monotonically. The word reorderings of the source sentence positions are restricted to the words of the German verb-group. For further details, see [15].

2.3 The Alignment Template Approach

The key element of the AT approach [10] are the *alignment templates*. These are pairs of source and target language phrases together with an alignment between the words within the phrases. The AT model decomposes the translation probability $Pr(f_1^J|e_1^I)$ by introducing two hidden variables: the sequence of alignment

templates z_1^K and the alignments within the templates $\tilde{a}_1^K$.

$$Pr(f_1^J|e_1^I) = \sum_{z_1^K, \tilde{a}_1^K} Pr(\tilde{a}_1^K|e_1^I) \cdot Pr(z_1^K|\tilde{a}_1^K, e_1^I) \cdot Pr(f_1^J|z_1^K, \tilde{a}_1^K, e_1^I) \quad (6)$$

$$= \sum_{z_1^K, \tilde{a}_1^K} \prod_{k=1}^{K} p(\tilde{a}_k|\tilde{a}_{k-1}) \cdot p(z_k|\tilde{e}_k) \cdot p(\tilde{f}_k|z_k, \tilde{e}_k) \quad (7)$$

There are three probability distributions:

- the phrase alignment probability $p(\tilde{a}_k|\tilde{a}_{k-1})$
- the probability of applying an alignment template $p(z_k|\tilde{e}_k)$
- the phrase translation probability $p(\tilde{f}_k|z_k, \tilde{e}_k)$

The AT approach uses a non-monotone search algorithm. The model scaling factors are trained with maximum entropy [9]. This is an extremely brief description of the AT model. For further details, see [10].

3 Phrase-Based Translation

3.1 Motivation

One major disadvantage of the single-word based (SWB) approach is that contextual information is not taken into account. As already said, the lexicon probabilities are based only on single words. For many words, the translation depends heavily on the surrounding words. In the SWB translation, this disambiguation is done completely by the language model. Often the language model is not capable of doing this. An example is shown in Fig. 3.

SOURCE	was halten Sie vom Hotel Gewandhaus ?
TARGET	what do you think about the hotel Gewandhaus ?
SWB	what do you from the hotel Gewandhaus ?
PBT	what do you think of hotel Gewandhaus ?

Fig. 3. Translation example

One way to incorporate the context into the translation model is to learn translations for whole phrases instead of single words. Here, a phrase is simply a sequence of words. So the basic idea of phrase-based translation (PBT) is to segment the given source sentence into phrases, then to translate each phrase and finally compose the target sentence from these phrase translations as seen in Fig. 4. As seen in the last phrase pair of the example, punctuation marks are treated as normal words.

SOURCE: abends würde ich gerne entspannen und vielleicht in die Sauna gehen .	
source segmentation	translation
abends	in the evening
würde ich gerne entspannen	I would like to relax
und	and
vielleicht in die Sauna gehen	maybe go to the sauna
.	.
TARGET: in the evening I would like to relax and maybe go to the sauna .	

Fig. 4. Example for phrase based translation

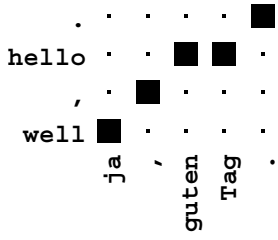


Fig. 5. Word aligned sentence pair

source phrase	target phrase
ja	well
ja,	well,
ja, guten Tag	well, hello
ja, guten Tag.	well, hello.
,	,
, guten Tag	, hello
, guten Tag.	, hello.
guten Tag	hello
guten Tag.	hello.
.	.

Fig. 6. Extracted bilingual phrases

3.2 Bilingual Phrases

Basically, a bilingual phrase is a pair of m source words and n target words. For extraction from a bilingual word aligned training corpus, we pose two additional constraints:

1. the words are consecutive and
2. they are consistent with the word alignment matrix.

This consistency means that the m source words are aligned only to the n target words and vice versa. The following criterion defines the set of bilingual phrases $\mathcal{BP}$ of the sentence pair $(f_1^J; e_1^I)$ that is consistent with the word alignment matrix A:

$$\mathcal{BP}(f_1^J, e_1^I, A) = \left\{ \left(f_j^{j+m}, e_i^{i+n} \right) : \forall (i', j') \in A : j \leq j' \leq j+m \leftrightarrow i \leq i' \leq i+n \right\}$$

This criterion is identical to the alignment template criterion described in [10]. Figure 5 is an example of a word aligned sentence pair. Figure 6 shows the bilingual phrases extracted from this sentence pair according to the defined criterion.

The extraction of the bilingual phrases from a bilingual word aligned training corpus can be done in a straightforward way. The algorithm in Fig. 7 computes the set $\mathcal{BP}$ with the assumption that the alignment is a function $A : \{1, ..., J\} \rightarrow \{1, ..., I\}$. It can be easily extended to deal with general alignments.

INPUT: f_1^J, e_1^I, A

FOR $i_2 = 1$ TO I DO

  FOR $i_1 = 1$ TO i_2 DO

    $SP = \{j | \exists i : i_1 \leq i \leq i_2 \wedge A(j) = i\}$

    IF consec(SP) THEN

      $j_1 = \min\{SP\}$

      $j_2 = \max\{SP\}$

      $\mathcal{BP} = \mathcal{BP} \cup \{(f_{j_1}^{j_2}, e_{i_1}^{i_2})\}$

OUTPUT: $\mathcal{BP}$

Fig. 7. Algorithm `extract-BP` for extracting bilingual phrases

3.3 Phrase-Based Translation Model

To use the bilingual phrases in the translation model we introduce the hidden variable B. This is a segmentation of the sentence pair $(f_1^J; e_1^I)$ into K phrases $(\tilde{f}_1^K; \tilde{e}_1^K)$. We use a one-to-one phrase alignment, i.e. one source phrase is translated by exactly one target phrase. So, we obtain:

$$Pr(f_1^J | e_1^I) = \sum_B Pr(f_1^J, B | e_1^I) \tag{8}$$

$$= \sum_B Pr(B | e_1^I) \cdot Pr(f_1^J | B, e_1^I) \tag{9}$$

$$= \alpha(e_1^I) \cdot \sum_B Pr(\tilde{f}_1^K | \tilde{e}_1^K) \tag{10}$$

Here, we assume that all segmentations have the same probability $\alpha(e_1^I)$. Next, we allow only monotone translations. This will result in a very efficient search. So the phrase $\tilde{f}_1$ is produced by $\tilde{e}_1$, the phrase $\tilde{f}_2$ is produced by $\tilde{e}_2$, and so on.

$$Pr(\tilde{f}_1^K | \tilde{e}_1^K) = \prod_{k=1}^{K} p(\tilde{f}_k | \tilde{e}_k) \tag{11}$$

Finally, we have to estimate the phrase translation probabilities $p(\tilde{f}|\tilde{e})$. This is done via relative frequencies:

$$p(\tilde{f}|\tilde{e}) = \frac{N(\tilde{f}, \tilde{e})}{N(\tilde{e})} \tag{12}$$

Here $N(\tilde{e})$ is the count of the phrase $\tilde{e}$. $N(\tilde{f}, \tilde{e})$ denotes the count of the event that $\tilde{f}$ has been seen as a translation of $\tilde{e}$. If one occurrence of $\tilde{e}$ has $N > 1$ possible translations, each of them contributes to $N(\tilde{f}, \tilde{e})$ with $1/N$. These counts are

calculated from the training corpus. If during the test an unknown word occurs, which was not seen in the training, this word is translated by itself.

Using a bigram language model and assuming Bayes decision rule (2), we obtain the following search criterion:

$$\hat{e}_1^I = \underset{e_1^I}{\operatorname{argmax}} \left\{ Pr(e_1^I) \cdot Pr(f_1^J | e_1^I) \right\} \tag{13}$$

$$= \underset{e_1^I}{\operatorname{argmax}} \left\{ \prod_{i=1}^{I} p(e_i|e_{i-1}) \cdot \alpha(e_1^I) \cdot \sum_B \prod_{k=1}^{K} p(\tilde{f}_k|\tilde{e}_k) \right\} \tag{14}$$

$$\approx \underset{e_1^I}{\operatorname{argmax}} \left\{ \prod_{i=1}^{I} p(e_i|e_{i-1}) \cdot \sum_B \prod_{k=1}^{K} p(\tilde{f}_k|\tilde{e}_k)^\lambda \right\} \tag{15}$$

In the last step, we omitted the segmentation probability $\alpha(e_1^I)$. We also introduced the translation model scaling factor λ [14]. Using the maximum approximation for the sum over all segmentations, we obtain the following search criterion:

$$\hat{e}_1^I \approx \underset{e_1^I, B}{\operatorname{argmax}} \left\{ \prod_{i=1}^{I} p(e_i|e_{i-1}) \cdot \prod_{k=1}^{K} p(\tilde{f}_k|\tilde{e}_k)^\lambda \right\} \tag{16}$$

3.4 Monotone Search

The monotone search can be efficiently computed with dynamic programming. For the maximization problem in (16), we define the quantity $Q(j, e)$ as the maximum probability of a phrase sequence that ends with the word e and covers positions 1 to j of the source sentence. $Q(J+1, \$)$ is the probability of the optimal translation. The \$ symbol is the sentence boundary marker. When finishing a hypothesis, we have to apply the conditional probability $p(\$|e')$, which denotes the probability of the sentence end after the word e'. We obtain the following dynamic programming recursion:

$$Q(0, \$) = 1 \tag{17}$$

$$Q(j, e) = \max_{\substack{0 \le j' < j, \\ e', \tilde{e}}} Q(j', e') \cdot p(f_{j'+1}^{j}|\tilde{e})^\lambda \cdot p(\tilde{e}|e') \tag{18}$$

$$Q(J+1, \$) = \max_{e'} Q(J, e') \cdot p(\$|e') \tag{19}$$

During the search, we store back-pointers to the maximizing arguments. So after performing the search, we can generate the optimal translation. This method will be later referred to as **MonMax**. The resulting algorithm has a worst-case complexity of $O(J^2 \cdot V_e \cdot |\{\tilde{e}\}|)$. Here, V_e denotes the vocabulary size of the target language and $|\{\tilde{e}\}|$ denotes the number of target language phrases. Using efficient data structures and taking into account that not all possible target language phrases can occur in translating a specific source language sentence, we can perform a very efficient search.

For the search criterion in (15), we define the quantity $Q(j, e_1^i)$ as the maximum probability of a phrase sequence that results in the word sequence e_1^i and covers the positions 1 to j of the source sentence. $Q(J+1, \hat{e}_1^I)$ is the probability of the optimal translation $\hat{e}_1^I$.

$$Q(0, \$) = 1 \tag{20}$$

$$Q(j, e_1^i) = \sum_{\substack{0 \leq j' < j \\ 0 \leq i' < i}} Q(j', e_1^{i'}) \cdot p(f_{j'+1}^j | e_{i'+1}^i)^\lambda \cdot p(e_{i'+1}^i | e_{i'}) \tag{21}$$

$$Q(J+1, \hat{e}_1^I) = \max_{e_1^I} Q(J, e_1^I) \cdot p(\$ | e_I) \tag{22}$$

This method will be later referred to as **MonSum**. The resulting algorithm has a worst-case complexity of $O(J^2 \cdot {V_e}^I \cdot |\{\tilde{e}\}|)$. Because of the sum criterion it is not allowed to apply language model recombination. This results in the factor ${V_e}^I$. In most statistical translation systems the maximum approximation is used, e.g. [3,5,10,18], but we will show in Sect. 4 that the sum criterion yields better results. These monotone algorithms are especially usefull for language pairs that have a similar word order, e.g. Spanish-Catalan or Italian-English.

3.5 Non-monotone Search

An analysis of the monotone translation results for the language pair German-English shows that many translation errors are due to the monotony constraint. Therefore in this section, we describe a way to extend the search described above to allow non-monotone translations. The idea is to use a reordering graph (RG) to restrict the number of possible word orders.

Reordering Graph. A RG is a directed acyclic graph with one start node and one goal node. The nodes are numbered from 0 (start) to N (goal). The numbering must be consistent with a topological order of the graph. Each node is marked with a coverage vector. This is a bit vector b of size J (the source sentence length) with the property $b[j] = 1$ iff the source position j is already covered, i.e. translated. The RG has the additional property that for each node its coverage vector differs from the coverage vector of each predecessor by exactly one bit. The start and the goal node are marked with 0^J and 1^J. We define $Pred(n)$ as the set of all predecessors (direct and indirect) of the node n. We define $S(n_1, n_2)$ as the source words covered by n_2 but not by n_1 in the same order as in the source sentence.

We gain a RG by removing non-needed information from a word graph generated by the SWB search and combining equivalent nodes, i.e. nodes with the same coverage vector.

Search. The search on the RG can be done by dynamic programming. The idea is similar to the monotone search, but instead of going over all source

positions j, we go over all nodes n of the RG from 0 to N. When visiting a node the topological sorting guarantees that all its predecessors have already been processed. Using maximum approximation, the quantity $Q(n, e)$ is defined as the maximum probability of a phrase sequence ending with the word e and translating $S(0, n)$.
We obtain the following dynamic programming recursion:

$$Q(0, \$) = 1 \tag{23}$$

$$Q(n, e) = \max_{\substack{n' \in Pred(n), \\ e', \tilde{e}}} Q(n', e') \cdot p(S(n', n)|\tilde{e})^{\lambda} \cdot p(\tilde{e}|e') \tag{24}$$

$$Q(N+1, \$) = \max_{e'} Q(N, e') \cdot p(\$|e') \tag{25}$$

This method will be later referred to as **ExtMax**. The equations for the sum criterion are analog. We define the quantity $Q(n, e_1^i)$ as the maximum probability of a phrase sequence that results in the word sequence e_1^i and translating $S(0, n)$. The method using the sum criterion will be later referred to as **ExtSum**.
We obtain the following dynamic programming recursion:

$$Q(0, \$) = 1 \tag{26}$$

$$Q(n, e_1^i) = \sum_{\substack{n' \in Pred(n), \\ 0 \leq i' < i}} Q(n', e_1^{i'}) \cdot p(S(n', n)|e_{i'+1}^i)^{\lambda} \cdot p(e_{i'+1}^i|e_{i'}) \tag{27}$$

$$Q(N+1, \hat{e}_1^I) = \max_{e_1^I} Q(N, e_1^I) \cdot p(\$|e_I) \tag{28}$$

3.6 Pruning

To further speed up the search and reduce the memory requirements, we apply threshold and histogram pruning. Note that with applying these pruning techniques the sum criterion is only approximately fulfilled. This is because if a hypothesis is pruned away, further contributions of extensions of this hypothesis are lost.

Threshold Pruning. The idea of threshold pruning is to remove all hypotheses that have a low probability relative to the best hypothesis. We need a threshold pruning parameter q, with $0 \leq q \leq 1$. We define $Q_0(j)$ as the maximum probability of all hypotheses for a source sentence position j. We prune a hypothesis iff:

$$Q(j, e_1^i) < q \cdot Q_0(j)$$

Histogram Pruning. The idea of histogram pruning is to restrict the maximum number of hypotheses for each source sentence position. So, only a fixed number of the best hypotheses is kept.

4 Results

4.1 Corpora

We present results on the VERBMOBIL task, which is a speech translation task in the domain of appointment scheduling, travel planning, and hotel reservation [17]. Table 1 shows the corpus statistics of this task. We use a training corpus, which is used to train the translation model and the language model, a development corpus, which is used to estimate the model scaling factors, and a test corpus.

Table 1. Characteristics of training corpus (Train, PM=punctuation marks), manual lexicon (Lex), development corpus (Dev), test corpus (Test)

		No Preprocessing		With Preprocessing	
		German	English	German	English
Train	Sentences	58 073			
	Words incl. PM	519 523	549 921	522 933	548 874
	Words excl. PM	418 979	453 632	421 689	456 629
	Singletons	3 453	1 698	3 570	1 763
	Vocabulary	7 940	4 673	8 102	4 780
Lex	Entries	12 779			
	Extended Vocabulary	11 501	6 867	11 904	7 089
Dev	Sentences	276			
	Words	3 159	3 438	3 172	3 445
	Trigram PP	-	28.1	-	26.3
Test	Sentences	251			
	Words	2 628	2 871	2 640	2 862
	Trigram PP	-	30.5	-	29.9

4.2 Criteria

So far, in machine translation research does not exist one generally accepted criterion for the evaluation of the experimental results. Therefore, we use a large variety of different criteria. In all experiments, we use the following error criteria:

- WER (word error rate):
  The WER is computed as the minimum number of substitution, insertion and deletion operations that have to be performed to convert the generated sentence into the target sentence. This performance criterion is widely used in speech recognition.
- PER (position-independent word error rate):
  A shortcoming of the WER is the fact that it requires a perfect word order. The word order of an acceptable sentence can be different from that of the

target sentence, so that the WER measure alone could be misleading. To overcome this problem, we introduce as additional measure the PER. This measure compares the words in the two sentences ignoring the word order.
- mWER (multi-reference word error rate):
  For each test sentence, there is not only used a single reference translation, as for the WER, but a whole set of reference translations. For each translation hypothesis, the edit distance (number of substitutions, deletions and insertions) to the most similar sentence is calculated [8].
- BLEU score:
  This score measures the precision of unigrams, bigrams, trigrams and fourgrams with respect to a whole set of reference translations with a penalty for too short sentences [11]. Unlike all other evaluation criteria used here, BLEU measures accuracy, i.e. the opposite of error rate. Hence, large BLEU scores are better.
- SSER (subjective sentence error rate):
  For a more detailed analysis, subjective judgments by test persons are necessary. Each translated sentence was judged by a human examiner according to an error scale from 0.0 to 1.0. A score of 0.0 means that the translation is semantically and syntactically correct, a score of 0.5 means that a sentence is semantically correct but syntactically wrong and a score of 1.0 means that the sentence is semantically wrong [8].
- IER (information item error rate):
  The test sentences are segmented into information items; for each of them, the translation candidates are assigned either "OK" or an error class. If the intended information is conveyed and there are no syntactic errors, the sentence is counted as correct [8].
- ISER (information item semantic error rate):
  This criterion is like the IER, but does not take into account slight syntactic errors.

4.3 PBT Variants

Table 2 shows the results for the PBT variants presented in this paper. As one may expect, the non-monotone variant yields better results than the monotone one. For the sum criterion, there is an improvement of the SSER of 7.8% absolute, which is 19.6% relative. We conclude that a for German-English translation non-monotone search is important to obtain good translation results. Typically in statistical translation systems the maximum approximation is used. Because of the simplicity of the presented PBT model, the sum over all segmentations can be carried out. Using the sum criterion instead of the maximum approximation improves translation quality. For the monotone search, there is an improvement of the SSER of 1.1% and for the non-monotone search of 0.6%.

4.4 Comparison with Other Systems

We compare the PBT results to the two other statistical translation systems, namely the SWB approach (see Sect. 2.2 and [15]) and the AT approach (see

Table 2. Comparison of different PBT variants

System	Variant	WER	PER	mWER	BLEU	SSER	IER	ISER
PBT	MonMax	46.9	33.3	42.0	42.1	40.8	40.2	19.0
	MonSum	45.9	32.2	40.9	43.4	39.7	40.0	19.2
PBT	ExtMax	42.5	30.4	36.7	49.5	32.5	33.0	17.7
	ExtSum	42.3	30.1	36.3	50.0	31.9	31.9	16.8

Table 3. Comparison of different translation systems

System	Variant	WER	PER	mWER	BLEU	SSER	IER	ISER
SWB	MON	49.0	35.2	43.4	37.0	47.0	51.7	33.2
	GE	41.9	31.4	35.9	47.5	35.1	39.0	21.6
PBT	MonSum	45.9	32.2	40.9	43.4	39.7	40.0	19.2
	ExtSum	42.3	30.1	36.3	50.0	31.9	31.9	16.8
AT		39.2	29.3	33.1	51.1	30.5	33.9	17.4

Sect. 2.3 and [10]). Some translation examples are shown in Table 4. In [2,13] an example-based approach is mentioned that is to some extend similar to PBT. The results are not included because they are evaluated on a different test set and therefore not comparable.

As Table 3 shows, the monotone PBT outperforms by far the monotone SWB translation. There is an improvement of the SSER of 7.3% absolute, which is 15.5% relative. The non-monotone PBT yields better results than the non-monotone SWB variant. There is an improvement of the SSER of 3.2%. So, this rather simple and straightforward phrase-based model performs better than the more complicated SWB model. We conclude that incorporating the local context into the translation model is important to achieve good translation results. One way to do this is the use of bilingual phrases.

On the other hand, PBT does not reach the performance of the AT approach, which is still 1.4% better. A possible reason is the generalization capability of the AT approach.

5 Conclusion

In this paper, we have presented a statistical translation model, which is based on bilingual phrases. Compared to the two other statistical approaches, this is a rather simple method, which results in a very efficient dynamic programming search algorithm. In the result section, we have compared this model to the SWB and AT approaches.

The major conclusions are:

1. Using bilingual phrases instead of single words in the translation model significantly improves translation quality.

Table 4. Translation examples

SOURCE	wollen wir am Abend Essen gehen ?
TARGET	would you like to go out for a meal in the evening ?
PBT MON	we will want evening go out to eat ?
PBT EXT	do we want to go out to eat in the evening ?

SOURCE	ich würde am dreißigsten gern mit dem Zug fahren .
TARGET	I would like to take the train on the thirtieth .
PBT MON	I would thirtieth like to go by train .
PBT EXT	on the thirtieth I would like to go by train .

SOURCE	dann müssen wir noch die Rückreise klären .
TARGET	then we still have to arrange the return journey .
PBT MON	then we still have to the return trip clarify .
PBT EXT	then we still have to clarify the return trip .

SOURCE	am Mittwoch fahren wir mit dem Zug wieder zurück nach Hamburg .
TARGET	on Wednesday we will go back by train to Hamburg .
SWB GE	on Wednesday we go by train from Hamburg again .
PBT MON	on Wednesday we go by train again back to Hamburg .

SOURCE	das Flugzeug ist dann um zwölf Uhr fünfundzwanzig in Hannover .
TARGET	the plane will arrive at Hanover at twenty-five past twelve .
SWB GE	the flight is at eleven twenty five in Hanover .
PBT MON	the plane is at twelve twenty-five in Hanover .

SOURCE	ich buche in dem Königshotel zwei Einzelzimmer mit Dusche .
TARGET	I will book two single rooms with a shower at the Königshotel .
SWB GE	I will book the Königshotel two singles with shower .
PBT MON	I will book in the Königshotel two single rooms with shower .
PBT EXT	I will book two single rooms in the Königshotel with shower .

2. For translating from German to English a non-monotone search is essential to produce good translations.
3. The sum criterion performs better than the maximum approximation.

Further investigations will concern the segmentation probability $Pr(B|e_1^I)$, which has so far been omitted. The use of hierarchical phrases, e.g. the pattern pairs in [2], might be interesting. Smoothing the phrase probabilities could result in additional improvements.

References

1. Auerswald, M.: Example-based machine translation with templates. [17] 418–427
2. Block, H.U.: Example-based incremental synchronous interpretation. [17] 411–417

3. Brown, P.F., Della Pietra, S.A., Della Pietra, V.J., Mercer, R.L.: The mathematics of statistical machine translation: Parameter estimation. Computational Linguistics **19** (1993) 263–311
4. Emele, M.C., Dorna, M., Lüdeling, A., Zinsmeister, H., Rohrer, C.: Semantic-based transfer. [17] 359–376
5. Germann, U., Jahr, M., Knight, K., Marcu, D., Yamada, K.: Fast decoding and optimal decoding for machine translation. In: Proc. of the 39th Annual Meeting of the Association for Computational Linguistics (ACL), Toulouse, France (2001) 228–235
6. Manning, C.D., Schütze, H.: Foundations of statistical natural language processing. MIT Press, Cambridge, MA (1999)
7. Ney, H., Nießen, S., Och, F.J., Sawaf, H., Tillmann, C., Vogel, S.: Algorithms for statistical translation of spoken language. IEEE Trans. on Speech and Audio Processing **8** (2000) 24–36
8. Nießen, S., Och, F.J., Leusch, G., Ney, H.: An evaluation tool for machine translation: Fast evaluation for MT research. In: Proc. of the Second Int. Conf. on Language Resources and Evaluation (LREC), Athens, Greece (2000) 39–45
9. Och, F.J., Ney, H.: Discriminative training and maximum entropy models for statistical machine translation. In: Proc. of the 40th Annual Meeting of the Association for Computational Linguistics (ACL). (2002) 8 pages To appear.
10. Och, F.J., Tillmann, C., Ney, H.: Improved alignment models for statistical machine translation. In: Proc. of the Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora, University of Maryland, College Park, MD (1999) 20–28
11. Papineni, K.A., Roukos, S., Ward, T., Zhu, W.J.: Bleu: a method for automatic evaluation of machine translation. Technical Report RC22176 (W0109-022), IBM Research Division, Thomas J. Watson Research Center (2001)
12. Reithinger, N., Engel, R.: Robust content extraction for translation and dialog processing. [17] 428–437
13. Tessiore, L., v. Hahn, W.: Functional validation of a machine interpretation system: Verbmobil. [17] 611–631
14. Tillmann, C.: Word re-ordering and dynamic programming based search algorithms for statistical machine translation. PhD thesis, Computer Science Department, RWTH Aachen, Germany (2001)
15. Tillmann, C., Ney, H.: Word re-ordering and DP-based search in statistical machine translation. In: COLING '00: The 18th Int. Conf. on Computational Linguistics, Saarbrücken, Germany (2000) 850–856
16. Vogel, S., Ney, H., Tillmann, C.: HMM-based word alignment in statistical translation. In: COLING '96: The 16th Int. Conf. on Computational Linguistics, Copenhagen, Denmark (1996) 836–841
17. Wahlster, W., ed.: Verbmobil: Foundations of speech-to-speech translations. Springer Verlag, Berlin, Germany (2000)
18. Wang, Y.Y., Waibel, A.: Modeling with structures in statistical machine translation. In: COLING-ACL '98: 36th Annual Meeting of the Association for Computational Linguistics and 17th Int. Conf. on Computational Linguistics. Volume 2, Montreal, Canada (1998) 1357–1363

Compiling Dynamic Agent Conversations

Pierre Bonzon

HEC, University of Lausanne
1015 Lausanne, Switzerland
pierre.bonzon@unil.ch

Abstract. We consider defining executable dialogues for communicating agents. Towards this end, we introduce agent classes whose communication primitives are based on deduction. Their operational semantics are given by an abstract logical machine that is defined purely in sequential terms. These agents communicate under the control of plans requiring a synchronization flag. These plans can be rewritten as dialogues with an implicit synchronization. Reversibly, dialogues can be compiled back into plans and then executed on the sequential machine. Sub-dialogues can be entered from any dialogue, such achieving dynamic conversation structures.

1 Introduction

Communication in multi-agent systems can be exemplified using the FIPA proposal [3]. In this approach, a set of normative *communicative acts* (or messages) is first defined. They represent the buildings blocks of a dialogue between agents. At an aggregated level, agent *interaction protocols* define generic sequences of messages representing a complete conversation (or dialogue) between agents. This enables agents to anticipate each other response according to some *conversation patterns*. Although the FIPA specifications contain predefined protocols, they do not impose upon agent to follow these standards (i.e. agent developers can adopt their own protocols). Conversations that are defined in this way have a fixed structure that can be laid down using some kind of graphical representation. Formalisms that have been proposed for this purpose include *graphs* generated by deterministic finite automata [3], colored Petri *nets* [6], or UML sequence *diagrams* [7].

This approach can be qualified as *static*, in the sense that every possible move must be made explicit beforehand and expressed in terms of alternative sequences of communicating acts (with possible feedbacks or resume). In particular, there is no provision for *composing* existing protocols on demand e.g., it is not possible for a given protocol to call another protocol at run time, and thus define *dynamic* conversation structures. In many ways, this situation is truly reminiscent of the early days of computer programming i.e., at the time when programs were monolithic objects lacking decomposition into procedures, function or subprograms. These programs, just like FIPA protocols, were truly static objects, and there was no such thing as a stack of activation records reflecting the dynamic embedding of successive procedure calls.

We strongly believe that there is a definite need for dynamic conversation structures i.e., for a model of agent conversation that would include the embedding of successive and/or nested protocol calls within a multi-agent system state. First of all, a

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 33-47, 2002.

much higher degree of modularity and reusability could be thus achieved. Secondly, and perhaps more importantly, this should allow for an easier diagnosis and recovery from the various deadlocks that can occur in a conversation i.e., when one party fails to answer as expected. We have been looking therefore for a model of *dynamic conversation structures*. Furthermore, we wanted to come up with a *conversation language* that would lead to directly executable specifications.

As a prerequisite, a formal model of agent communication at the message level is required. Recently, Hendriks and al. have advocated a new approach based on synchronized pairs of communication primitives [5]. Defined as "neutral" actions enjoying a well-defined and clear semantics, these logical primitives can be used for many different purposes, including the implementation of speech acts. In order for two agents to communicate, both parties must first agree to an exchange (e.g., by independently using an external exchange protocol based on these synchronized pairs). Each exchange then involves either a deductive or an abductive task performed independently by one of the agents. As an example (that we develop later), this can be used in collaborative models to synchronize successive negotiation rounds as well as the successive steps in each round.

In order to get executable specifications, the operational semantics of the complete model must be given in sequential terms. Towards this end, we first extended a general model of individual agent with sensing [9] to include the notion of *non-deterministic plans* (or *nd-plans* in short). Originally given in abstract functional terms, this model was developed into a set of concrete procedures that represent a sequential abstract machine generating runs for non-deterministic agents. We then integrated within this framework a simplification of the communication model [5]. In the resulting model [1], two agents communicate under the control of nd-plans requiring each a synchronization flag. These plans are directly executable on our abstract machine (in essence, a sequential machine with deductive capabilities).

Our goal in this paper is to try and obtain executable specifications of agent dialogues that do not require an explicit synchronization. Towards this end, we shall consider a communication language that will be defined in terms of *branching sequences*. This language will allow us to rewrite nd-plans as *dialogues*. Reversibly, dialogues can be compiled back into nd-plans, and then executed on the original machine. Our main result is that dialogues can be *deterministically* and *simply sequentially* executed using a single pair of synchronization and sequencing flags. Furthermore, as sub-dialogues can be entered from any dialogue, we thus achieve dynamic conversation structures.

The rest of this paper is organized as follows: in section 2, we review the definition of deductive communicating agents. Section 3 proposes a language for writing agent dialogues that are equivalent to nd-plans. Section 4 gives its operational semantics via compiling functions for translating dialogues into nd-plans. This approach is finally illustrated with an example of a multi-round negotiation.

2 A Model of Deductive Communicating Agents

As a prerequisite for the interpretation and/or compilation of executable agent dialogues, we first need to define and implement a model of communicating agents. Towards this end, we used and simplify the proposal made by Hendriks and al. [5]. Following a purely logical approach, they introduced two pairs of neutral

lowing a purely logical approach, they introduced two pairs of neutral communication primitives i.e., *tell/ask* and *req/offer* that correspond to data exchanges enjoying a well-defined semantics and can be used for many different purposes. In each pair, *r* is designated as the *receiver* and *s* as the *sender.* The first pair can be defined as follows:

> Message *tell(r,□)* from sender *s* provides *r* with data □, and message *ask(s,□)* from receiver *r* expresses his willingness to solve a query □ using any data sent by *s*. Both messages are sent without reciprocal knowledge of what the other agent wants or does. In particular, the data □ volunteered by *s* is not given in response to *r*'s asking. If these two messages are put together through some kind of external handshake or synchronization, then by using both his own knowledge and the data □ told by *s*, receiver *r* will try and answer his query □. Formally, receiver *r* will compute the most general substitution □ such that
> $l^r \square \square \vdash \square\square$
> where l^r represents the local state of the receiver. As the unknown substitution □ stands after the deduction sign ⊢, i.e. within the conclusion, this amounts to a classical *deductive* task.

According to Hendricks & al., □ in ask*(s,□)* can contain free variables but □ in *tell(r,□)* must be closed; furthermore, $l^s \vdash \square$ is not required (i.e., *s* is not required to be truthful or honest). We shall illustrate this type of exchange through a simple example. Let the local state l^r of *r* be such that

$l^r \vdash$ *father(abram,isaac)* □ *father(isaac,jacob)*

and let us consider the following pair of messages

message sent by *s*: *tell(r,□XYZ father(X,Y)* □ *father(Y,Z)* □ *grandfather(X,Z))*
message sent by *r*: *ask(s, grandfather(X,jacob)).*

In this first scenario, *s* tells *r* a closed implication, and *r* asks *s* for some data that could allow him to find out who is the grandfather of *jacob.* Using the data sent by *s*, *r* is then able to deduce the substitution *X=abram.*

In contrast, the second pair can be defined as follows:

> Message *req(r,□)* from sender *s* requests *r* to solve query □, and message *offer(s,□)* from receiver *r* expresses his willingness to use data □ for solving any query submitted by *s.* When put together, these two messages will lead the receiver *r* to find the possible instantiations of his free variables in □ that allow him to deduce □. Formally, receiver *r* will compute the most general substitution □ such that
> $l^r \square \square\square \vdash \square$.
> As the unknown substitution □ stands before the deduction sign ⊢, i.e. within the premises, this amounts to a non-classical *abductive* task.

According to Hendricks & al., □ in *req(r,□)* must be closed but □ in *offer(s,□)* can contain free variables; furthermore, $l^r \vdash \square$ is not required, but $l^r \vdash \square\square$ is not allowed. To illustrate this second type of exchange, let us consider the following pair of messages

message sent by *s*: *req(r, □X grandfather(X,jacob))*
message sent by *r*: *offer(s, father(X,Y)* □ *father(Y,Z)* □ *grandfather(X,Z)).*

In this second scenario, *s* requests *r* to find out if there is a known grandfather for *jacob*. Independently, *r* offers *s* to abduce a substitution for the free variables in his □ that would allow him to answer. In this case, using his knowledge contained in l^r and the implication he offers, *r* can abduce the same substitution as before.

In both of the above exchanges, no data is sent back to *s*, and the corresponding formal semantics captures the processing done by *r* only. In other words, the sender will not be aware of the results of the receiver's computation. For the sender to get this result, a reversed exchange (e.g. an *ask/tell*) is needed. While this is perfectly appropriate for the first type of exchange (after-all, the sender who volunteers data is not necessarily interested in the receiver's computations), we feel that, in the second case, the sender who expresses a need for data should automatically benefit from the receiver's computations. Furthermore, as abductions are difficult to achieve and implement, we favor exchanges that do not rely on abduction. Giving up the *req/offer* pair, we thus defined and implemented instead a simplified *call/return* pair that relies on deduction only and makes the results of the receiver's computations available to the sender. By doing so, we did end up with a less powerful model. It is interesting to note however that all *req/offer* examples given in [5] can be expressed as *call/return* invocations. In particular, if the receiver's local state includes closed forms of his offer □, then a *req(r,□)/offer(s,□)* pair reduces to a *call/return* pair. This new pair is defined as follows:

> In the *call(r,□)/return (s,□)* pair, both □ and □ can contain free variables. This exchange is then interpreted as the sender *s* calling on *r* to instantiate the free variable in his query □. Independently, the receiver *r* is willing to match his query □ with the sender's □ and return the instantiations that hold in his own local state. Formally, receiver *r* will *deductively* compute the substitution □ s. t.
> □□ = □□ and l^r ⊢ □□.
> As indicated above, this information will be made available to the sender, i.e. substitution □ will be sent back to *s*.

To illustrate this, let us suppose that we now have

l^r ⊢ *father(abram,isaac)*□*father(isaac,jacob)*□
□*XYZ father(X,Y)* □ *father(Y,Z)* □ *grandfather(X,Z)*
message sent by *s*: *call(r, grandfather(X,jacob))*
message sent by *r*: *return(s, grandfather(X,Y))*.

This exchange is to be interpreted as *s* calling on *r* to find out the grandfather of *jacob* i.e., to instantiate the free variable in his query. Independently, *r* is willing to match the sender's call and to return the substitutions that hold in his local state. Once again the substitution *X=abram* will be found. In contrast to the previous exchanges however, this information will be sent back to the sender.

3 A Language for Agent Dialogues

In order to first get an intuitive feeling for the language we have implemented, we shall start with an illustrating example. Towards this end, let us consider a simplified version of the *two-agent meeting-scheduling* example of [5]. In this application, one

agent is designated as the *host* and the other one as the *invitee*. Both agents have free meeting slots e.g.,

$l^{host} \vdash meet(13) \square meet(15) \square meet(17)$
$l^{invitee} \vdash meet(14) \square meet(16) \square meet(17)$

and they must find their earliest common slot (in this case, *17*). We shall make use of a predicate *epmeet(T1,T)* meaning "*T1 is the earliest possible meeting time after T* ", defined as

$meet(T1) \square (T1>=T) \square \square (meet(T) \square (T>=T) \square (T<T1)) \square epmeet(T1,T).$

The solution involves successive negotiation *cycles*. The host has the responsibility of starting each cycle with a given lower time bound *T*. A cycle comprises three steps, each step involving a exchange of messages. In the first step, the host initializes a *call/return* exchange calling on the invitee to find out his earliest meeting slot *T1* after *T*. In the second step, roles are swapped: the invitee initializes a *call/return* calling on the host to find out his earliest meeting slot *T2* after *T1*. In the third step, the host either confirms an agreement on time *T2* (if *T1=T2*) by initializing a *tell/ask* exchange, or starts a new cycle with *T2* as his new lower bound.

This solution can be informally expressed as follows:

"start with a *call/return* exchange,
proceed with a *return/call* exchange,
conclude with a *tell/ask* exchange or *resume"*

A formal rewriting under the form of two synchronized dialogues is then

dialog*(invite(Invitee, T), [T1,T2],*
*[***call***(Invitee,epmeet(T1,T)),*
return*(Invitee,epmeet(T2,T1)),*
*((T1=T2 | [***tell***(Invitee,confirm(T2)),*
execute*(save(meeting(T2)))]);*
*(T1\=T2| [***resume***(invite(Invitee,T2))]))])*

dialog*(reply(Host),[T,T1,T2],*
*[***return***(Host,epmeet(T1,T)),*
call*(Host,epmeet(T2,T1)),*
*((T1=T2 | [***ask***(Host,confirm(T2)),*
execute*(save(meeting(T2)))]);*
*(T1\=T2| [***resume***(reply(Host))]))])*

where "," and ";" are sequence (or conjunctive) and alternative (or disjunctive) operators , respectively. Variables start with capital letters, and variables that are local to a dialogue are listed before the messages. As it is quite apparent in this example, each dialogue consists of a *branching sequence* of messages i.e., a *sequence* with an *end alternative.* Similarly to lists, branching sequences can have an embedded structure. Unless they are resumed (with a *resume* message), dialogues are exited at the end of each embedded branching sequence. Actions interleaved with messages can be executed with an *execute* message. Although this simple example does not make use of this possibility (for a more complex example, see section 5), sub-dialogues can be entered (with an *enter* message) from any dialogue, such achieving *dynamic conversation* structures. The corresponding BNF syntax is given below in Fig. 1

<dialog> ::= **dialog**(<dialogName>(<dialogParams>),<varList>,<branchSeq>)

<varList> ::= [] || [<varName>|<varList>]

<branchSeq> ::= [] || [<alt>] || <seq>

<alt> ::= <guardMes> || (<guardMes>;<alt>)

<seq> ::= [<mes>|<branchSeq>]

<guardMes> ::= (<guard>|<branchSeq>)

<mes> ::= <messageName>(<messageParams>)

<messageName> ::= **ask** || **tell** || **call** || **return** || **execute** || **enter** || **resume**

Fig. 1. BNF productions

As usual, "|" separates the head and tail of a list i.e., $[m_1|[m_2|...[]]]=[m_1,m_2,...]$. We also use "|" to isolate the guard in a guarded message. To avoid confusion, we use "||" as metasymbol for representing choices. We leave out the definitions for *names, parameters* and *guards*, these being identifiers, first order terms and expressions, respectively. Branching sequences permit end alternatives, but do not allow for starting or middle alternatives i.e., cannot contain the list pattern *[<alt>|<branchSeq>]*.

4 Compiling Dialogues into nd-Plans

In order to define the operational semantics of our language, we shall first implement a multi-agent system under the form of an *abstract machine*. In this system, agents using the primitive messages defined in Sect. 2 will communicate under the control of non-deterministic plans using a synchronization flag. These plans are equivalent to dialogues with an implicit synchronization. Reversibly, dialogues can be compiled back into plans and then executed on the sequential machine. As it is common, we shall not distinguish here (at least until the end of Sect. 6) between the term *dialogue* and *conversation.*

4.1 An Abstract Machine for Communicating Agents with Plans

Let us consider a multiagent system consisting of a *class* of identical agents. Similarly to classical *objects*, we shall distinguish the *class* itself, considered as an object of type "*agent class*", and its class *members* i.e., the objects of type "*agent instance*". The class itself will be used both as a repository for the common properties of its members (including the definition of the abstract machine), and as a blackboard for agent communication. We shall consider purely communicating agents i.e., the environment will be ignored, and thus there will be no agent *sensing*. We will also delay the dynamic embedding of plans until we introduce the compilation of dialogues. There will thus be no need yet to consider local variables, or to maintain stacks of activation records. The *local state* of a class of agents will be defined by a vector $l = [l^{Class}, l^1 ... l^n]$, where the components l^{Class} and l^i are the local state of the class and its

members identified by an integer $i=1…n$, respectively. We will make use of a predicate *agent* and assume that $l^{Class} \vdash agent(i)$ whenever agent i belongs to the class.

Messages exchanged between class members must use a data transport system. We shall abstract this transport system as follows: any message sent by an agent (this message being necessarily half of an exchange between a sender and a receiver, as introduced in the previous section, with the exception of the *resume* message to be used to reenter the same plan) will be first posted in the class. The class itself will then interpret the message's contents, wait for the second half of the exchange (thus achieving synchronization), and finally perform the computation on behalf of the receiver. Each message will be *blocking* until the exchange's completion i.e., no other exchange of the same type will be allowed between the sender and the receiver before the exchange is completed.

Let us assume that the language defining each l^i includes a set $P = \{p_1, p_2, …\}$ of non-deterministic plan names (*nd-plan* in short) and four predicates *plan, priority, do* and *switch*. For each agent *i,* its current plan $p^i \in P$ refers to a set of implications "*conditions*" $\Rightarrow$ *do*(p^i, a), where a is an action. In the case of communicating agents, actions will be identified with messages, and conditions will include a synchronization flag *sync* referring to the successful execution of the preceding message. As an example, let us consider the plans corresponding to the dialogues of section 3. In order for the first cycle in each plan to be started, we will assume that flags *sync(dialog(invite(Invitee,T)))* and *sync(dialog(reply(Host)))* have been raised beforehand. The corresponding host and invitee plans i.e., *invite(Invitee,T))* and *reply(Host))*, are defined as follows

sync(dialog(invite(Invitee, T))) $\Rightarrow$
do(invite(Invitee,T), **call***(Invitee,epmeet(T1,T)))*

sync(call(Invitee,epmeet(T1,T))) $\Rightarrow$
do(invite(Invitee,T), **return***(Invitee,epmeet(T2,T1)))*

sync(return(Invitee,epmeet(T2,T1))) $\wedge$ *T=T2* $\Rightarrow$
do(invite(Invitee,T), **tell***(Invitee,confirm(T2)))*

sync(return(Invitee,epmeet(T2,T1))) $\wedge \neg$*(T=T2)* $\Rightarrow$
do(invite(Invitee,T), **resume***(invite(Invitee,T2)))*

sync(dialog(reply(Host))) $\Rightarrow$
do(reply(Host), **return***(Host,epmeet(T1,T)))*

sync(return(Host,epmeet(T1,T))) $\Rightarrow$
do(reply(Host), **call***(Host,epmeet(T2,T1)))*

sync(call(Host,epmeet(T2,T1))) $\wedge$ *T=T2* $\Rightarrow$
do(reply(Host), **ask***(Host,confirm(T2)))*

sync(call(Host,epmeet(T2,T1))) $\wedge \neg$*(T=T2)* $\Rightarrow$
do(reply(Host), **resume***(reply(Host)))*

Message *resume*, used by an agent to reenter a plan, is interpreted by a state transformer function τ^i defined as

$\tau^i([l^{Class},… l^i,…],$ **resume(p)**$) = [l^{Class}, …l^i$ *-{plan(_),sync(_)}* $\cup$ *{plan(p),sync(dialog(p))},…]*

Similarly, *processes* of priority n encompass implications "*conditions*" $\Rightarrow$ *do(n, a)*. Let us further assume that each agent's initial nd-plan p_0^i and the class highest priority n_0 can be deduced from l, i.e. $l^i \vdash plan(p_0^i)$ and $l^{Class} \vdash priority(n_0)$. The abstract ma-

chine that defines the run of a class as a loop interleaving individual agent run cycles is then defined by the following procedure

procedure $run^{Class}(l)$
loop for all i ***such that*** $l^{Class} \vdash agent(i)$ ***do***
  if $l^i \vdash plan(p_0^i)$
then $react^i(l,p_0^i)$;
if $l^{Class} \vdash priority(n_0)$
then $process^{Class}(l,n_0)$

In each cycle, initial plans p_0^i are activated by a procedure $react^i$. Synchronization occurs though a procedure $process^{Class}$. These procedures are defined as:

procedure $react^i(l,p^i)$
if $l^i \vdash do(p^i, a)$
then $l \square \square^+(l,a)$
else if $l^i \vdash switch(p^i, p'^i)$
then $react^i(l,p'^i)$

procedure $process^{Class}(l,n)$
if $l^{Class} \vdash do(n, a)$
then $l \square \square^{Class}(l,a)$
else if $n > 0$
then $process^{Class}(l,n-1)$

In each $react^i$ call, the agent's first priority is to carry out an action a from its current plan p^i. Otherwise, it may switch from p^i to p'^i, a recursive call to $react^i$ leading in turn to the same options. In any case, the next run cycle will again deduce and activate (possibly different) initial plans p_0^i. If the *switch* predicate defines directed acyclic graphs rooted at each possible p_0^i, corresponding thus to a hierarchy of plans with decreasing priorities and thus ensuring termination, then $react^i$ will always select the applicable nd-plan that has the highest *implicit* priority. This feature allows directing an agent to adopt a new plan whenever a certain condition occurs. It is however not required to implement purely communicative agents and the ***else*** branch of *react* will thus be ignored in the sequel. Similarly, $process^{Class}$ will execute the process that has the highest *explicit* priority.

The state transformer function $\square^s$ used to interpret message $tell(r, \square)$ sent by s is

$\square^s([l^{Class},\ldots l^s,\ldots], \mathbf{tell(r,\square)}) =$
if $busy(tell(r,\square)) \square l^s$
then $[l^{Class} \square \{ack(s,tell(r,\square))\},\ldots\ l^s \square \{busy(tell(r,\square))\},\ldots]$
else $[l^{Class},\ldots\ l^s,\ldots]$

The functions for messages $ask(s,\square)$, $call(r,\square)$ and $return(s,\square)$ are similarly defined. According to these functions, each message is thus first posted in the class, a *busy* flag is raised in the agent state, and the message waits to be synchronized. Synchronization occurs when two messages belonging to the same pair have been acknowledged. This synchronization is triggered by two *priority processes* defined as

$ack(s,tell(r,\square)) \square ack(r,ask(s,\square))) \square\ do(2, \mathbf{tellAsk(s,r,\square,\square)})$
$ack(s,call(r,\square)) \square ack(r,return(s,\square)) \square\ do(1, \mathbf{callReturn(s,r,\square,\square)})$

The state transformer function $\square^{Class}$ achieving synchronization is:

$\Box^{Class}([l^{Class},...l^s,...l^r,...],$ **tellAsk(s,r,□,□)**$) =$
if $l^r \Box \Box \vdash \Box\Box$
then $[l^{Class} - \{ack(s,tell(r,\Box)), ack(r,ask(s,\Box))\},...$
$l^s\text{-}\{busy(tell(r,\Box)), sync(\_)\} \Box \{sync(tell(r,\Box))\},...$
$l^r\text{-}\{busy(ask(s,\Box)), sync(\_)\} \Box \{sync(ask(s,\Box\Box))\},...]$
else $[l^{Class},... l^s,...l^r,...]$

$\Box^{Class}([l^{Class},...l^s,...l^r,...],$ **callReturn(s,r,□,□)**$) =$
if $\Box\Box = \Box\Box$ ***and*** $l^r \vdash \Box\Box$
then $[l^{Class} - \{ack(s,call(r,\Box)), ack(r,return(s,\Box))\},...$
$l^s\text{-}\{busy(call(r,\Box)), sync(\_)\} \Box \{sync(call(r,\Box\Box))\},...$
$l^r\text{-}\{busy(return(s,\Box)), sync(\_)\} \Box \{sync(return(s,\Box\Box))\},...]$
else $[l^{Class},... l^s,...l^r,...]$

In short, all the flags are removed and a new *sync* flag carrying the computation results is raised. To ensure a simple execution scheme, a single such synchronization flag is used at any time.

4.2 Compiling Dialogues

The concrete operational semantics for the complete language of fig. 1 are finally given below in fig. 2. This definition takes the form of compiling functions for translating dialogues into nd-plans. It closely follows the BNF syntax given above, with an *exit* message being automatically added at the end of each branching sequence. Each compiled message is assigned a unique *sequence* number. This number is used to define the sequencing flag *seq(D(I))* that will be raised when message with sequence number *I* from dialogue *D* is executed. This flag in turn will be used as a *sequencing condition* for the next message (recall that our abstract machine does not have a program counter, and that execution is triggered by deduction). Sequencing conditions are required to serialize successive messages that may have the same synchronizing conditions and thus could otherwise not be distinguished (recall also that synchronization occurs when the two messages belonging to a primitive pair have been acknowledged: two distinct messages whose preceding pairs are identical will thus have the same synchronizing condition). As an alternative solution, the sequence number could be introduced in the synchronization flag.

As stated by the implication compiled by $comp_{mes}$, the condition for the execution of message *P(X)* is *var(Var)□Sync□seq(D(I)),* where *Sync* is its synchronizing condition, *seq(D(I))* its sequencing condition (with *I* referring to the preceding message), and *Var* is the list of local variables from the current dialogue. When this condition is checked, the variables in the list *Var* will be unified with the corresponding variables in *Sync*. Before *P(X)* is actually executed, the sequencing condition for the next message will be updated into *seq(D(J+1))*, with *J+1* referring to the current message. The local variables will be similarly updated. As possible instantiations will be carried over from *Sync,* this will allow for the result of the previous message to be taken into account.

$comp_{dialog}(dialog(D,Var,\mathbf{BranchSeq})) = \{dialog(D,Var)\} \square comp_{branchSeq}(D,Var,\mathbf{BranchSeq},sync(dialog(D)),0,0,N)$

$comp_{branchSeq}(D,Var,\mathbf{[]},Sync,I,J,J+1) = \{var(Var) \square Sync \square seq(D(I)) \square do(D,save(seq(D(J+1)))) \square do(D,save(var(Var))) \square do(D,\mathbf{exit(D)})\}$

$comp_{branchSeq}(D,Var,\mathbf{[Alt]},Sync,I,J,N) = comp_{alt}(D,Var,\mathbf{Alt},Sync,I,J,N)$

$comp_{branchSeq}(D,Var,\mathbf{Seq},Sync,I,J,N) = comp_{seq}(D,Var,\mathbf{Seq},Sync,I,J,N)$

$comp_{alt}(D,Var,\mathbf{GuardMes},Sync,I,J,N) = comp_{guardMes}(D,Var,\mathbf{GuardMes},Sync,I,J,N)$

$comp_{alt}(D,Var,\mathbf{(GuardMes;Alt)},Sync,I,J,N) = comp_{guardMes}(D,Var,\mathbf{GuardMes},Sync,I,J,K) \square comp_{alt}(D,Var,\mathbf{Alt},Sync,I,K,N)$

$comp_{seq}(D,Var,\mathbf{[Mes|BranchSeq]},Sync,I,J,N) = comp_{mes}(D,Var,\mathbf{Mes},Sync,I,J,K) \square comp_{branchSeq}(D,Var,\mathbf{BranchSeq},sync(\mathbf{Mes}),K,K,N)$

$comp_{guardMes}(D,Var,\mathbf{(Guard|BranchSeq)},Sync,I,J,N) = comp_{branchSeq}(D,Var,\mathbf{BranchSeq},Sync \square \mathbf{Guard},I,J,N)$

$comp_{mes}(D,Var,\mathbf{P(X)},Sync,I,J,J+1) = \mathbf{if}\ P \square \{\mathbf{tell},\mathbf{ask},\mathbf{call},\mathbf{return},\mathbf{execute},\mathbf{enter},\mathbf{resume}\}\ \mathbf{then}\ \{var(Var) \square Sync \square seq(D(I)) \square do(D,save(seq(D(J+1)))) \square do(D,save(var(Var))) \square do(D,\mathbf{P(X)})\}$

Fig. 2. Compiling functions

The last argument of each compiling function returns the last sequence number assigned by the function. Two input arguments i.e., *I* and *J*, provide the sequence numbers that are required to compile the sequencing conditions for the current and next messages. When compiling end alternatives in function $comp_{alt}$, *I* keeps its value while *J* is set to *K*, the current last sequence number. When compiling sequences in function $comp_{seq}$, both *I* and *J* are set to *K* (globally, both *I* and *J* work similarly to the split second hand of chronograph i.e., they eventually fly back to *K,* but under different conditions).

When compiling end alternatives, function $comp_{alt}$ similarly keeps its synchronization flag *Sync.* In contrast, when compiling sequences *[Mes|BranchSeq],* function $comp_{seq}$ introduces a new synchronization flag *sync(Mes)*. The exclusion of starting or middle alternatives in branching sequences precludes the compilation of complex synchronization flags of the form $sync(Sync_1) \square sync(Sync_2) \square \ldots$ that otherwise would propagate in parallel and then possibly lead to backtracking on execution. Similar remarks apply for sequencing flags i.e., like pure sequences, branching sequences can be serialized using a single sequencing flag. In other words, this means that messages may have at most one direct predecessor message. But, contrary to pure sequences, they may have multiple successors. A *simple sequential* execution scheme can thus be achieved by updating a single pair of *sync* and *seq* flags at each step. We have the following result, whose proof intuitively follows from the preceding remarks:

Proposition Dialogues based on branching sequences can be *simply sequentially* executed (i.e. by using a single pair of synchronization and sequencing flags). Furthermore, if all the guards in a given alternative are mutually exclusive, then this execution will be *deterministic*.

Turning now to the interpretation of primitive messages (recall that *tell, ask, call,* and *return* have been defined earlier), we have the following new state transition functions, where a stack t^i is used to store agent's i current active dialogue embedding:

$\sigma^i([\ldots<l^i,t^i>,\ldots],$ **execute**$(a)) =$
if $\sigma^i([\ldots<l^i,t^i>,\ldots], a) = [\ldots<l^i,\tau^i>,\ldots]$
then $[\ldots<l^i,\tau^i\cup\{sync(\_)\}\cup\{sync(execute(a))\},t^i>,\ldots]$
else $[\ldots<l^i,t^i>,\ldots]$

$\sigma^i([\ldots\tau^i,t^i,\ldots],$ **resume**$(q)) =$
if $dialog(q,v)\in\tau^i$
then $[\ldots\tau^i\text{-}\{plan(\_),var(\_),seq(\_),sync(\_)\}$
$\cup\{plan(q),var(v),seq(q(0)),sync(dialog(q))\},t^i,\ldots]$
else $\{undefined\}$

$\sigma^i([\ldots\tau^i,t^i,\ldots],$ **enter**$(q)) =$
if $dialog(q,v)\in\tau^i$
then if $\{plan(p),var(w),seq(p(s))\}\in\tau^i$
then $[\ldots\tau^i\text{-}\{plan(\_),var(\_),seq(\_),sync(\_)\}$
$\cup\{plan(q),var(v),seq(q(0)),sync(dialog(q))\},$
push$(t^i,\{p,w,s\}),\ldots]$
else $[\ldots\tau^i\text{-}\{sync(\_)\}\cup\{plan(q),var(v),seq(q(0)),sync(dialog(q))\},t^i,\ldots]$
else $\{undefined\}$

$\sigma^i([\ldots\tau^i,t^i,\ldots],$ **exit**$(q)) =$
if not empty(t^i) ***and top***$(t^i) = \{p,w,s\}$
then $[\ldots\tau^i\text{-}\{plan(\_),var(\_),seq(\_),sync(\_)\}\cup\{plan(p),var(w),seq(p(s)),sync(enter(q))\},$
pop$(t^i),\ldots]$
else $[\ldots\tau^i\text{-}\{plan(\_),var(\_),seq(\_),sync(\_)\},t^i,\ldots]$

Contrary to *entered* dialogues, *resumed* dialogues are not stacked. They can thus be used to implement *reentrant monitors* (see below in section 6).

5 Example: A Multi-round Negotiation

As an example of the complete language, let us consider the extension to n agents of the meeting-scheduling problem. This solution involves successive *rounds*, each round involving in turn successive *cycles*. The *host* dialogue initializes each *round*, directly followed by a general agreement or recursively by a new round. In each round, the host will *tell* in turn each of the invitees (again through recursion) to *reply* and then *invite* him for a negotiation *cycle.* Each such cycle will be initialized with the bilateral agreement just reached between the host and the previous invitee. The round itself will end up with a tentative proposal handed over to the main host dialogue.

In their *guest* dialogue, invitees will *ask* for instructions and then either *reply* and recursively *ask* for new instructions, or *ask* for the general agreement. In contrast to the solution involving only two agents, the agreement phase (i.e., *ask* for confirmation and then *save* the information) cannot directly follow a bilateral agreement, and thus is not included in the *invite/reply* but in the *host/guest* dialogues. The dialogues are:

```
dialog(host(Att,T), [T1],
 [enter(round(Att,T,T1)),
 ((T=T1 | [enter(tellAll(Att,turn(end))),
 enter(tellAll(Att,confirm(T1))),
 execute(save(meeting(T1)))]);
 (T\=T1|[enter(host(Att,T1))]))])
dialog(round([Att1|AttR],T,T2), [T1],
 [tell(Att1,turn(reply)),
 enter(invite(Att1,T,T1)),
 ((AttR=[] | [execute(equals(T2,T1))]);
 (AttR\=[] | [enter(round(AttR,T1,T2))]))])
dialog(tellAll([Att1|AttR],C), [],
 [tell(Att1,C),
 ((AttR=[] | []);
 (AttR\=[] | [enter(tellAll(AttR,C))]))])
dialog(invite(Invitee,T,T3), [T1,T2],
 [call(Invitee,epmeet(T1,T)),
 return(Invitee,epmeet(T2,T1)),
 ((T1=T2 | [execute(equals(T3,T2))]);
 (T1\=T2 | [enter(invite(Invitee,T2,T3))]))])
dialog(guest(Host),[Turn,T],
 [ask(Host,turn(Turn)),
 ((Turn=reply |[enter(reply(Host)),
 enter(guest(Host))]);
 (Turn=end | [ask(Host,confirm(T)),
 execute(save(meeting(T)))]))])
dialog(reply(Host), [T,T1,T2],
 [return(Host,epmeet(T1,T)),
 call(Host,epmeet(T2,T1)),
 ((T1=T2 | []);
 (T1\=T2 | [enter(reply(Host))]))])
```

6 Conclusion and Possible Extensions

We proposed a formal language for modeling dynamic agent conversation. The corresponding operational semantics is given by compiling functions that maps dialogues onto non-deterministic plans executable on a sequential abstract machine. The underlying protocol for the exchange of information relies on a flag mechanism to ensure synchronization. One could object that this specification is too low level. We might well try and describe it in a more abstract way, for example by using a transition semantics as done in [5]. We would however be left short of true executable specifications. A number of useful extensions to the basic model are reviewed below.

6.1 Recovering from Failures

A *simple failure* to answer (because the deduction involved in a communication primitive did not succeed) or a synchronization that did not occur (because the expected agent was not available, did not anticipate the request, or simply failed) are examples of deadlocks that could prevent dialogues to proceed as expected. We have already implemented a mechanism for *catching* a simple failure to answer. By default, this failure will propagate through embedded dialogues via a forced *exit.* It can be caught on demand in the first calling dialogue where it can be appropriately processed. Timeouts could be similarly handled. As an example, let us consider below the following extension for the *host* dialogue, in which a successful and an unsuccessful deduction lead to catch a status equal to *end* and *fail*, respectively.

dialog*(host(Att,T), [T1,Status],*
*[***enter***(round(Att,T,T1)|***catch***(Status)),*
*((Status=end |[((T=T1 | [***enter***(tellAll(Att,turn(quit))),*
execute(save(meeting(T1)))]);*
*(T\=T1 |[***enter***(host(Att,T1))]))]);*
*(Status=fail |[***enter***(tellAll(Att,turn(quit))),*
execute(save(failed(meeting)))]))])*

6.2 Monitoring External Commands

The overall behavior of any agent should allow for entering any dialogue on demand. This behavior could be defined as a reentrant dialogue monitoring external interrupts:

dialog*(monitoring(I), [Act,P,X],*
*[***ask***(I, command(Act(P(|X)))),*
*((Act=enter |[***enter***(P(|X)),*
resume(monitoring(I))]);*
*(Act=execute|[***execute***(P(|X)),*
resume(monitoring(I)))]))])*

Each agent *i* would have to be associated with a *sensing* procedure implemented as

procedure *sense*^{i}*(l)*
if *"the interrupt handler coupled with i receives the command Act(P(|X)))"*
then *l* □ □^{i}*(l,* **tell***(i, command(Act(P(|X)))))*

Calls to *sense*^{i} could then be interleaved with calls to *react*^{i} within each *run* cycle.

In this implementation, each agent *i* will thus first engage in a *tell(i,*□*)/ask(i,*□*)* "auto exchange" with its sensing procedure. After retrieving a command i.e., after □*=*□*=command(Act(P(|X))),* it will then either *enter* a dialogue or *execute* an action, and then resume its monitoring task.

6.3 Engaging in Multiple Conversations

Agents should be allowed to engage in multiple conversations. Instead of providing a conversation language with a parallel (or concurrency) operator that could be used at

the message level i.e., to interleave possible concurrent messages (as put forward in the languages *3APL* [5] and *ConGolog* [4], among others), we favor the simpler solution whereby each agent is a multithreaded entity interleaving concurrent conversations.

Just as a multi-agent system was implemented as a multi-threaded entity of agents using predicate *agent*, a multi-threaded agent can be implemented within an extended abstract machine using an additional predicate *conversation* as follows

procedure $run^{Class}(l)$
loop for all *i* ***such that*** $l^{Class} \vdash agent(i)$ ***do***
$sense^{i}(l)$;
for all *j* ***such that*** $l^{i} \vdash conversation(j)$ ***do***
if $l^{ij} \vdash plan(p_0^{ij})$
then $react^{ij}(l, p_0^{ij})$;
etc …

A new primitive message *concurrent* could then be used by any dialogue (such as the monitoring dialogue itself) to create a new conversation thread when required i.e., we would then have

dialog*(monitoring(I), [Act,P,X],*
*[***ask***(I, command(Act(P(|X)))),*
*((Act=enter |[***concurrent***(P(|X)),*
resume*(monitoring(I))]);*
etc …

In this new implementation, dialogues are considered as syntactic entities that can be attached to multiple conversations implemented as independent threads. To ensure consistency of this extended formalism, the monitoring dialogue itself must be attached to each agent's initial conversation thread.

7 Related Work

The subject of modeling agent conversation is relatively new. Apart form the work already mentioned in the introduction i.e., [3] [6] and [7], which concentrates on defining graphical frameworks for representing static conversation patterns, earlier contributions include [2] and [8]. None of this work however seems to address the issue of modeling dynamic conversation structures. Furthermore, as the underlying communication models are either left unspecified or made to rely on speech acts, there is no simple ways to define the corresponding operational semantics leading to directly executable specifications. Finally, if conversations are to be used for modeling the *social ability* of agents, then (as already argued by Hendricks and al [5]) computational equivalents for speech acts should not be included in an agent communication language, as done in KQML or FIPA ACL. Communications primitives should instead be kept neutral, and *mental attitudes* should be allowed to *emerge* eventually as intelligent behavior.

References

1. P. Bonzon, An Abstract Machine for Communicating Agents Based on Deduction, in: J.-J. Meyer & M.Tambe (eds), *Intelligent Agents VIII*, LNAI vol. 2333, Springer Verlag (2002)
2. R. Elio, A. Haddadi and A. Singh, Task Models, Intentions and Agent Conversation Policies, in: *Proc. PRICAI-2000*, LNAI vol. 1886, Springer Verlag (2000)
3. FIPA Specifications, available on line at http://www.fipa.org
4. G. de Giacomo, Y.Lespérance and H. Levesque, ConGolog, a Concurrent Programming Language Based on the Situation Calculus, *Artificial Intelligence*, vol. 121 (2000)
5. K.V. Hendricks, F.S. de Boer, W.van der Hoek and J.-J. Meyer, Semantics of Communicating Agents Based on Deduction and Abduction, *Proceedings IJCAI99 Workshop on ACL* (also Utrecht University Technical Report UU-CS-1999-09)
6. M. Nowostawski, M. Purvis and S. Cranefield, A Layered Approach for Modelling Agent Conversations, in: E T. Wagner and O.F. Rana (eds), *Proc. 2nd Int. Workshop on Infrastructure for Agents, MAS, and Scalable MAS, 5th Int. Conf. on Autonomous Agents,* Montreal (2001)
7. J.J. Odell, H.V.D. Parunak and B. Bauer, Representing Agent Interaction Protocols in UML, in: P. Ciancarini and M. Wooldridge (eds), *Agent-Oriented Software Engineering*, Springer Verlag (2001)
8. R. Scott, Y. Chen, T. Finin, Y. Labrou and Y. Peng, Modeling Agent Conversations with Colored Petri Nets, in: *Working Notes of the Worshop on Specifying and Implementing Conversation Policies, 3rd Int. Conf. On Autonomous Agents,* Seattle (1999)
9. M. Wooldridge and A. Lomuscio, Reasoning about Visibility, Perception and Knowledge, in: N.R. Jennings and Y. Lespérance (eds*), Intelligent Agents VI*, LNAI vol. 1757 , Springer Verlag (2000)

Dynamic Pricing of Information Products Based on Reinforcement Learning: A Yield-Management Approach

Michael Schwind and Oliver Wendt

Chair of Economics, esp. Information Systems
Frankfurt University, D-60054 Frankfurt, Germany
schwind,wendt@wiwi.uni-frankfurt.de
http://www.wi-frankfurt.de

Abstract. Pricing of information services gains an increasing importance in an IT environment, which is characterized by more and more decentralized computing resources (e.g. P-2-P computing). Even if pricing theory represents a kernel domain of economic research the pricing problem related to automated information production processes could not be handled satisfactory. This stems from the combination of high fixed costs with negligible variable costs. Especially in airline industries this problem is addressed by heuristics in the so called "*Yield Management*" (*YM*) domain. The paper presented here, shows the transferability of these methods to the information production and services domain. Pricing a bundle of complementary resources can not be solved by the simple addition of value functions. Therefore we introduce *Machine Learning* (*ML*) techniques to master complexity. *Artificial Neural Networks* (*ANN*) are used for the joint representation of the multidimensional value functions and *Genetic Algorithms* (*GA*) should help train them in a first effort. While this does not lead to outstanding results, we try *Reinforcement Learning* (*RL*) in a second approach. This *ML* method provides encouraging results for efficient adaptive pricing of resource attribution related to the multidimensional *YM* problem.

1 Introduction

Coordination of economic activities, at first requires efficient attribution of scarce resources. Using the price as a measure for their relative shortness is a widely accepted principle in economics. Further on, classic microeconomic theory [9, 22,12] implicates that goods do not loose their utility until consumption takes place. However, this is not the case for a class of goods, which can be considered as perishable. Pricing of durable goods dominates the marketing literature [25, 31] up to now, but there can't be found much effort in pricing perishable goods to achieve an efficient resource allocation in this domain. *Yield Management* (*YM*) tries to close this gap by employing a dynamic-pricing or a contingent allocation approach in theory as well as in practice and provides valuable results in the airline domain [8] since years. In the following we demonstrate, that the

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 51–66, 2002.

problem of optimal information pricing has many parallels to the *YM* task and sketch how to transfer suitable solution methods to the *Information Production and Information Services* (*IPIS*) domain.

2 Yield-Management Problems

The failure of conventional pricing theory in many industrial sectors stems mostly from the existence of limiting factors, which can be expanded timely only at the price of prohibitive high costs. Production processes however, which are characterized by high costs for capacity resourcing and dynamic standby, have mostly low variable marginal costs for an additional unit of output [4]. This leads to high contribution margins (*CMs*). Because each additional demand unit, bearing a positive *CM*, leads to a coverage of the irreversible predisposed costs for capacity resourcing, an appropriate price/quantity management provides a reasonable potential for profit rising.

Belobaba and Vogel [3,35] subsume all techniques of integrated pricing and capacity control, which maximize the fixed costs coverage by attributing the right capacity type to the right customer type to the *YM* domain. Kimes [16] identifies the following characteristics making production processes eligible for *YM* methods:

- Fixed production capacities
- Possibility of market segmentation
- Non-storability and perishableness of product units
- Pre-production sales
- High volatility of demand
- Low marginal costs of additional product units within capacity constraints / High costs for capacity expansion

Examples for perishable products can be found in food-, fashion-, hotel- and airline industry. Even if first approaches for overbooking management exist since the beginning of the 70s, a systematic development of *YM* methods started foremost with deregulation of American aviation industry in 1979 and was transferred to the hotel, travel, logistics and transportation sector in the late 80s [32, 36]. Since the end of the 90s applications for resource allocation in telecommunications are increasingly developed grounding on *YM* [15]. Perishableness in connection with services does not refer to the resource itself (rental car, hotel room, airplane seat), but the possibility of using it to generate returns in the actual period. The resource itself remains unchanged [36,28,6].

We will now consider the specifics of *YM* for ***Information Products and Information Services*** (*IPIS*): The definition of *IPIS* denotes the customized retrieval, replication or generation of data like e.g. the search and replication of MP3 files on a P-2-P Server, the risk evaluation of a portfolio by customer defined defaults or the calculation of a route for a logistic customer. Compared with physical services following differences can be recognized:

Processes of *IPIS* are often interruptible (preemptive); the state of processing can be buffered and the process resumed if resources are idle again. In the traditional *YM* domain this is not possible.

Whereas variable costs of production are low compared to fixed costs in the physical *YM* domain, they are negligible for *IPIS* applications.

Duration of *IPIS* tasks can't be predetermined as exactly as it is the case in the classic *YM* environment. The problem must be solved employing stochastic distributions for estimation.

3 Solving the Yield-Management Problem

Techniques to solve the *YM* problem, can be differentiated into heuristics and exact optimization procedures. The best known representatives of the first group are *Nested Booking Classes* (*NBC*) [14] and the concept of the *Expected Marginal Seat Revenue* (*EMSR*) [3]. Both concepts ground on the division of available seats into booking classes, which have to be sized before booking starts and adjusted during the booking process. Requests with higher *CMs* can make use of capacity from classes with lower *CMs*, if their contingent is used up. Whereas the determination of class limits is done by simulation of historic data in the *NBC* process, *EMSR* procedure detects the point, at which the *EMSR* of an additional capacity unit falls below the *EMSR* of the next lower class, because the probability of an amount of requests for the higher class is correspondingly lower.

Because *YM* represents a recurrent dynamic decision problem under uncertainty, static solutions like linear programming do not lead to an optimal solution [16]. Moreover *Stochastic Dynamic Programming* (*SDP*) has to be used to find exact solutions [1].

All *YM* decision problems can be described as a tree structure, where edges represent the individual decisions ($a \in A$ = *action space*) and vertices denominate the system states *(*$s \in S$ = *state space)*.

In our framework of decision process optimization the expected value of the next decision is required as additional parameter to describe the reward of an action ($r \in R$ = *reward function*):

$$\Re^{a}_{ss'} = E\left\{r_{t+1} \,|a_t = a\ , s_t = s, s_{t+1} = s'\right\} \tag{1}$$

In every node a decision about the selection of the next edge is taken. Additionally, one can assign a probability to the selection of the actions.

$$P^{a}_{ss'} = \Pr\left\{s_{t+1} = s' \,|s_t = s, a_t = a\right\} \tag{2}$$

A further parameter π_i is employed for this purpose. It is denoted as *decision policy*, because the strategy used to select the edges has direct impact on the probability of running through a particular path in the tree.

To facilitate the analytical handling of our decision problem while using π_i, the *Markov* property [27] is assumed. It implies, that the probability for the

state transition is independent of the previous system states and decisions. This avoids path dependency and enables a solution based on *SDP*. We now present an example for *SDP* applied to the time dependant *YM* problem:

We assume a contingent of 6 seats available in a small aircraft, flying from A to B. Seats can be sold single or as a bundle, implying that the remaining capacity can take the state values $S = \{0, 1, 2, 3, 4, 5, 6\}$.

Table 1. Exemplary distribution of booking-requests

Type	Probability	Requested Seats	Reward
F1	0.5	1 Seat	1 MU
F2	0.3	2 Seats	4 MU
F3	0.2	3 Seats	9 MU

The *YM* system receives capacity requests for this flight, which can be either accepted or rejected and differ in quantity of seats and price. For simplicity we assume only three types of seat requests (F1, F2 and F3) specified in Table 1. In addition we presuppose that the amount of incoming requests, each enumerated by a back counting index k, is known at decision time.

To achieve an optimal acceptance at stage k, a decision maker has to know the type of request k, the residual capacity i, the amount of the remaining $k-1$ requests for the case of rejection, the reward R_k in stage k and the value $V_{k-1}(i-f_k)$ of request $k-1$, for capacity i reduced by the required seat capacity f_k. It could then be tested, whether:

$$R_k + V_{k-1}(i - f_k) \geq V_{k-1}(i) \tag{3}$$

If Eq. (3) is true, the request will be accepted, otherwise rejected. The optimal policy for stage $k = 1$ can be calculated using the following instruction:

Accept any request, which is less or equal to the residual capacity, if there are no further incoming requests. The expected value $V_1(i)$ results from the remaining capacity i using the probability distribution given in Table 1. One can now determine the optimal decision policy for stage $k = 2$. The expected return on stage k =2 consists of the expected return $r(i, a)$ on this stage and the value of the remaining residual capacity $V_1(j)$ on stage $k = 1$, where j denotes the reduced residual capacity:

$$V_2(i) := \max_a [r(i, a) + \sum\nolimits_j p(i, a, j) V_1(j)] \tag{4}$$

By comparing the capacity value for all possible states i and each acceptance policy a, the optimal acceptance policy for each state i is evaluated. Thereby p denominates the transition probability from capacity i to capacity j under acceptance policy a. Iterating this process leads to an optimal *YM* policy for each stage k. Figure 1 shows the asymptotic progression of the maximum value

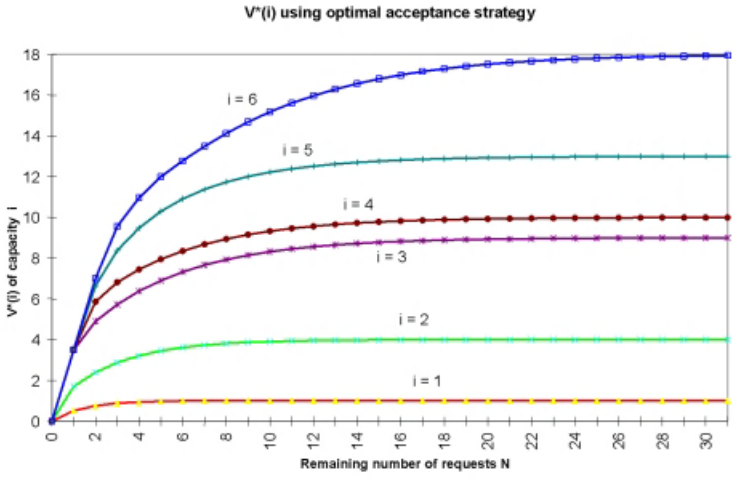


Fig. 1. Value of a given residual capacity vs. number of remaining requests

of a given residual capacity i subject to the amount N of remaining capacity requests.

Despite the optimality of the *SDP* approach, the computational complexity avoids widespread implementation. This applies in particular to the so called *Network Yield Management* (*NYM*) problem, where the customers do not only request the usage of one resource, but a resource bundle (e.g. a multi-leg flight or a hotel stay for several days). Approaches neglecting this combinatorial complexity, like the linear addition of several resource prices by means of a *Bid Pricing* (*BP*) procedure lead to suboptimal results [34]. Literature regarding complexity reduction in the *Markovian* state space takes only account of general clustering mechanisms and not of the multidimensional *NYM* problem [26,10].

3.1 Artificial Neural Networks for Value-Function Representation

A possible way to deal with the complexity problem related to the evaluation of the optimal value function V^* consists of the usage of *Artificial Neural Networks* (*ANN*) either to represent V^* or to map the decision function [13]. Both possibilities should be depicted brief.

In our context we employ a feed-forward *ANN* [7] using 3 layers of neurons. The following assignment of *ANN* layers seems to be advisable in our *YM* context:

Each resource available at valuation time t, represented as components of resource vector I, is mapped by a neuron in the input layer. Besides the k *resource-neurons*, a *time-neuron* is required in the input layer to feed the amount of remaining future requests into the *ANN*. On the output side a single output neuron indicates the value of the request (Fig. 2).

At request time the *ANN* determines the value of two different resource vectors I, namely of residual capacity in case of request-acceptance and residual capacity in case of request-rejection. The difference yields the *reservation price* for the request as decision criterion.

Employing this topology in connection with the back-propagation algorithm a satisfying approximation of V^* depicted in Fig. 1 can be achieved (Fig. 3).

Unfortunately it turns out to be difficult to use supervised learning in the *YM* domain, because the result of the decision process shows up foremost after the entry of the last request and the optimal total marginal contribution for the particular request profile is not known until it has been calculated employing *SDP*. One way to get out of this dilemma, is to train different *ANNs* with the same sample of a value function, calculated by *SDP* and selecting the best performing *ANNs*, hoping that the generalization property of the network will provide satisfying solutions measured by the *Average Marginal Contribution* (*AMC*).

According to this, the search for an optimal *ANN* representation of the value function can be seen as a parameter optimization problem. The *search space* is the vector space of all weights of *ANN* neurons, the goal function for a given parameter constellation is the *AMC* which is reached by the weights, if the particular *ANN* comes to application.

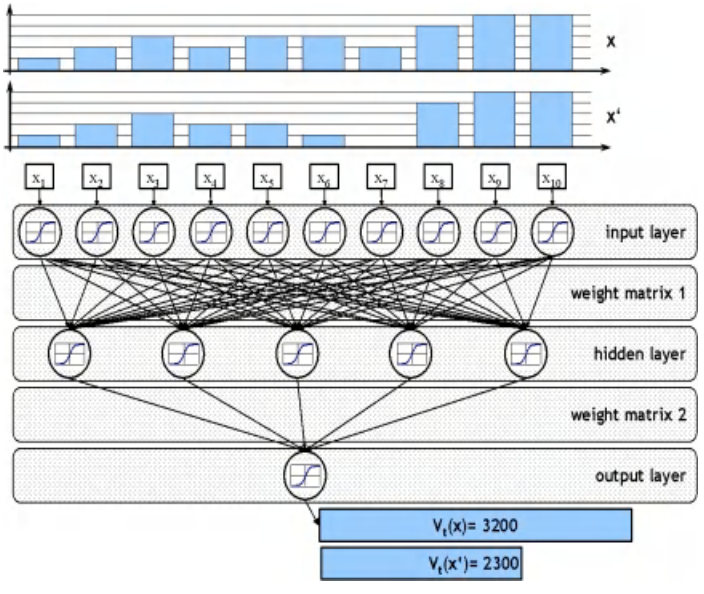


Fig. 2. Exemplary valuation of a reduction of resource capacity x5, x6 and x7 by a requested order; the expected *CM* amounts to 3200-2300=900 MU

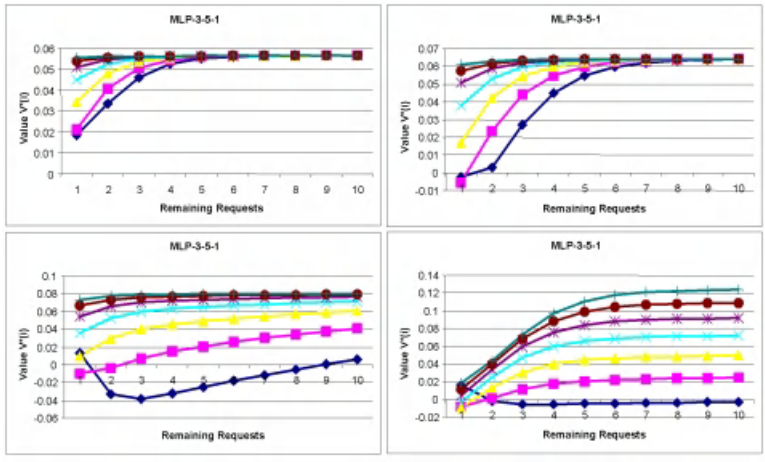


Fig. 3. Typical results of a back-propagation ANN 3-5-1 after 20.000, 40.000, 60.000 and 160.000 iterations

3.2 Using Evolutionary Techniques to Improve the Value-Function

Considering the meta problem of global parameter optimization, the selection of reasonable optimization strategies depends strongly on the structural properties of the search space, which are not known in our case.

In case of a unimodal space a gradient search could be applied. However it will hopelessly succumb to techniques based on *Evolution Strategies* [30] or *Genetic Algorithms* (*GAs*) [11]: These start searching at a multitude of points (here: individual *ANNs*) in search space and proceed with the evolutionary process by mutation, reproduction and selection.

The application of *GA*-based local search strategies raises the question of optimal sample size. Less simulations, carried out for a given *ANN* could lead to the repudiation of a superior vector of weights w in favor of an inferior vector w' due to estimation errors. Increasing the sample size helps reducing this risk but implies a lower number of parameterizations to be evaluated per time span.

Taking this tradeoff into account several *ANN* topologies were tested allocating two resources only. This allows the analytical calculation of the expected result using *SDP* and enables the rating of the *ANNs* function approximation performance.

This led to a desired overall selection probability for high-quality individuals. As *CO* operator a modified *1-point-crossover* was employed: For each layer, the rows of the weight matrices where cut at a randomly chosen column and recombined with the remaining lines respectively columns of the mating partner (see Fig. 4).

To determine the fitness of each particular *ANN* in a population 100 requests processes were sampled including 10 requests at a time. Finally the population

was sorted according to the fitness and reduced to the original size of 10 individuals.

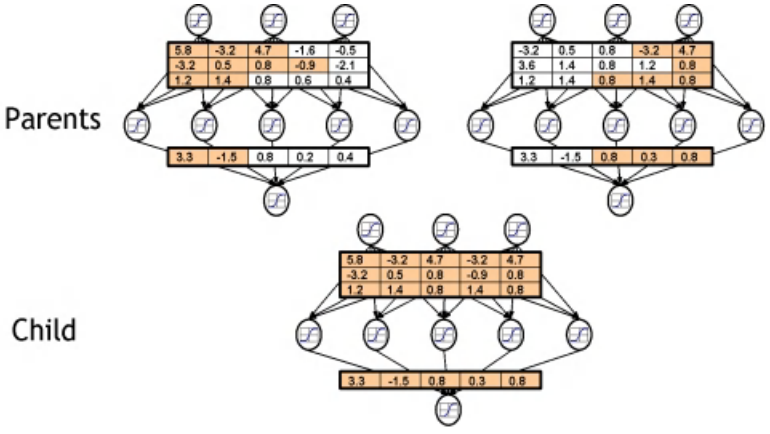


Fig. 4. Sample of a two ANN 1-point crossover

Figure 5 depicts the impact of population size on the average fitness of the value function evolving after 100 generations. The sampling rate denotes how many processes (10 requests each) are drawn from the expected demand distribution, to reduce the estimation error.

Each point represents the result of a *GA* after 100 generations. The plain is the smoothed mean value estimation of the average marginal contribution after 100 generations for each given population size and sampling rate. A positive effect of population size and sampling rate on the quality of the result can be recognized but may be misleading with respect to the necessary calculation effort.[1]

Employing simple *CO* and mutation operators used in this context, the computational expense lies mainly in the evaluation procedure. Therefore the following diagrams no longer show the performance for a fixed quantum of 100 generations, but a budget of 100.000 and 200.000 evaluations (Fig. 6): It turns out that best quality is achieved at small sampling rates and populations in the 100.000 evaluations study, whereas in the case of 200.000 evaluations larger populations come into vantage leaving the benefit of smaller sampling rates untouched. That means calculation time should be allocated into greater populations or a longer *GA* evaluation period, but not into a more precise determination of the objective function.

[1] The usage of population size 5 and sampling rate 10 per generation, implying 50 child-*ANNs* evaluated 5 times each, leads to 100*50*5=25.000 demand streams, whereas a population size of 20 leads to 100*200*100=2000.000 evaluations.

To demonstrate the scalability of our approach in the context of *NYM* a stochastic request sample generator for an arbitrary amount n of resource types was specified. For the sake of simplicity only 10 request samples for 10 different resource types where allowed. For each of the 10 request samples and each resource type a capacity requirement between 0 and 9 units is chosen randomly. The *CM* affected by the requests is defined randomly between 0 and 9 likewise.

Figure 7 shows the relative performance of the *GA-ANN* procedure compared to the optimal solution. The results indicate a declining performance for higher resource dimensions.

4 Reinforcement Learning for Yield-Management Processes

The fact, that domain specific knowledge about the structural properties of the value function (e.g. monotonicity) is not applied in the searching process and the quality of $V(s)$ can not be rated directly, leads to an increasing computational expense on higher resource dimensions. For this reason the application of an adaptive learning system, which replaces either the *ANN* representation of the value function or the *GA* procedure manipulating this representation seems to be indicated. *Reinforcement Learning (RL)*, a simulation based approach, which rests upon the *Bellman-SDP* seems to be a promising approach in this context [2,5].

4.1 Basic Idea of Reinforcement Learning

Considering the base elements of decision optimization –states, actions and reinforcements (*rewards*)– under a system-theoretic aspect the following model could be assumed besides the interpretation as a decision tree [33]:

- The *RL-agent* is connected to its *environment* via sensors.
- In every step of interaction the agent receives a *feedback* about the *state of environment* s_{t+1} and the *reward* r_{t+1} of its latest action a_t.
- The agent chooses an action a_{t+1} representing the output function, which changes state s_{t+2} of environment.
- The agent anew gets a feedback, through *reward* r_{t+2}.
- Objective of the agent is to optimize the *sum of rewards*.

4.2 Stochastic Dynamic Programming

We now introduce a framework for *Bellman-SDP* which enables us to deduct the *RL* algorithm used in our *YM* application. For this purpose, we introduce two new terms:

The *state value* $V^{\pi}(s)$ of a policy π is defined to be the expected value, which results from the discounted state value $V^{\pi}(s')$ of the next state s' summed up

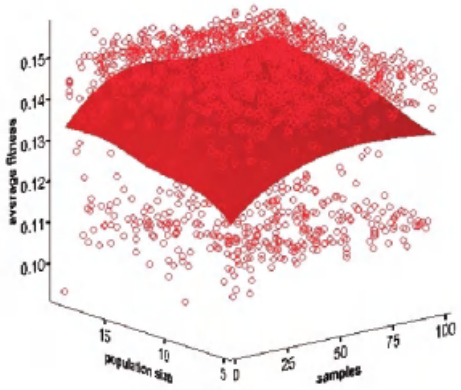


Fig. 5. Impact of population size and sampling rate on the quality of the value function (100 generations)

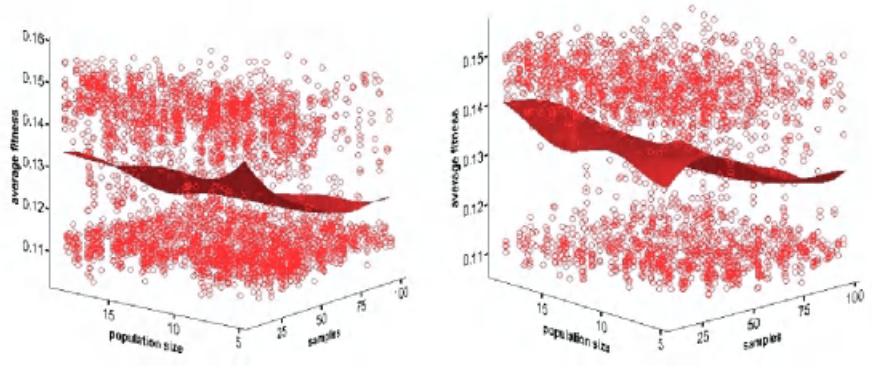


Fig. 6. Impact of population size and sampling rate on the quality of the value function (100.000 and 200.000 evaluations)

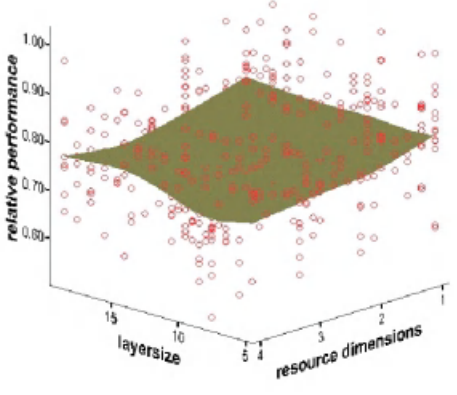


Fig. 7. Relative performance of the GA vs. layer size and amount of resources

with the expected reward r in $t+1$:

$$V^{\pi}(s) \;=\; E_{\pi}\left\{\, R_t |\, s_t = s \,\right\} \;=\; E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} | s_t \;=\; s \right\} \tag{5}$$

The *action value* $Q^{\pi}(s,a)$ of a policy π in addition, depends on the action chosen in s. Actions a which are not selected do not account for the calculation of $Q^{\pi}(s,a)$.

$$Q^{\pi}(s,a) \;=\; E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} |\, s_t \;=\; s,\, a_t \;= a \right\} \tag{6}$$

To find a path through the decision tree which maximizes the sum of rewards we need a definition of the optimal state value and the optimal action value respectively:

The *optimal state value* can be calculated recursively, based on the *Bellman* equation (Eq. (5)). In contrast to the normal state value not all decision alternatives are taken into account, but only the one which provides the highest value, by summing up the discounted optimal state value of the lower decision stage and the reward $r(a,s)$ of the optimal action a.

$$V^{*}(s) \;=\; \max_{a} \{ r(s,a) \;+\; \gamma V^{*}(s') \} \tag{7}$$

In analogy to Eq. (7) the *optimal action value* $Q^{*}(s,a)$ of a policy π can be written as:

$$Q^{*}\,(s,a) \;=\; E\;\left\{\, r_{t+1} \;+\; \gamma V^{*}(s_{t+1}) |\, s_t \;=\; s,\, a_t \;=\; a \right\} \tag{8}$$

The computation of $V^{*}(s)$, done by calculating $V_{k+1}(s)$ repeatedly, is known as *value iteration*. Alternatively one can think of a *policy adjustment* during the evaluation to avoid a complete decision tree exploration to rate a policy. Such a technique, called *policy iteration*, is a repeated process of *evaluation* and *improvement*: after optimizing a policy π yielding the improved policy π' a new value function $V^{\pi'}(s)$ can be calculated to improve π'.

4.3 Monte-Carlo Methods

A downside of using the *Bellman-SDP* lies in the necessity to pass all states of the decision tree to calculate the value function (*full backup method*). Because it is mostly sufficient to coarsely estimate $V(s)$, a *Monte-Carlo (MC) method* can be employed. Then it is not necessary to traverse the whole decision tree, but only several traces (episodes). With *MC* method the median values of the rewards are used as an estimator for $V(s)$. Two fundamental proceedings have been established in this context:

– *First-visit MC method:* The episodes are passed through, while recording the mean value of the *previous rewards in each visited node*. If a state s, which has already been visited occurs the mean value $V(s)$ recorded at the *first visit*, remains preserved.

– *Every-visit MC method:* This method differs from the first visit *MC* method only by the usage of an *update rule*, e.g. the *constant-α MC method*, to *improve the state value* of an already visited state s.

$$V^{\pi}_{neu}(s_t) = V(s_t) + \alpha\,[R_t - V(s_t)] \tag{9}$$

The *selection strategy* for the episodes is crucial with respect to the estimation of $V(s)$: Starting with an *arbitrary policy* an episode is generated in the *evaluation phase*. Along the trace of this episode action values are generated using the first- or every-visit method. Thereupon the *improvement phase* follows, yielding an optimization of the selected policy. This process is iterated in analogy to policy iteration. The *random strategy* triggers a *learning effect*, because *exploration* does take place.

4.4 Temporal-Difference Learning

While *SDP* makes it possible to select an optimal action a for the next state s on a decision level k, as far as all actions in the tree have been evaluated up to time t (*full backup*), the *MC* method depends on the evaluation of at least one episode, containing state s to calculate the Q-values and to take the optimal decisions. *Temporal Difference*(TD) learning combines the advantages of both methods. Beginning with the constant-α *MC* method the horizon of an episode can be shortened to one decision step $TD(0)$, while convergence is granted for small α:

$$V^{\pi}(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{10}$$

Up to now only the evaluation of a policy using the *TD*-method has been regarded, nothing has been said with respect to *policy improvement*. The same *alternation between policy evaluation and improvement* in the action space employed for the *MC* method, can be applied to the $TD(0)$ procedure. Writing Eq. (10) using Q-values enables us to formulate a policy iteration algorithm for the $TD(0)$ method.

```
Initialize Q(s,a) arbitrary
   Repeat for each episode
       Initialize s
       Select a from s by using a policy derived from Q
       Repeat (for each step of the episode):
           Perform action a and observe r, s'
           Select a'from s' using a policy derived from Q
            Q(s,a) ← Q(s,a) + α [r + γQ(s',a') − Q(s,a)]
            s ← s'; a ← a';
        until s is terminal state
```

Temporal-Difference Algorithm

4.5 Adapting Reinforcement Learning to Yield Management Problems

First, we substituted *SDP's* value iteration approach by a $TD(0)$ learning. Choosing an appropriate learning rate α turns out to be crucial in our setting: chosen too high, the system overreacts to single requests, chosen too low, we have too slow an adaptation process as soon as a significant change in demand occurs.

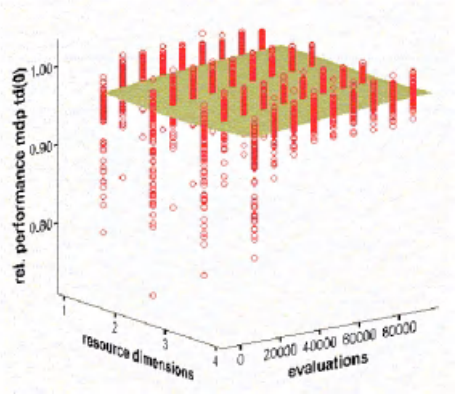


Fig. 8. Relative Performance of $TD(0)$ Reinforcement Learning as a function of the number of resource dimensions and evaluations

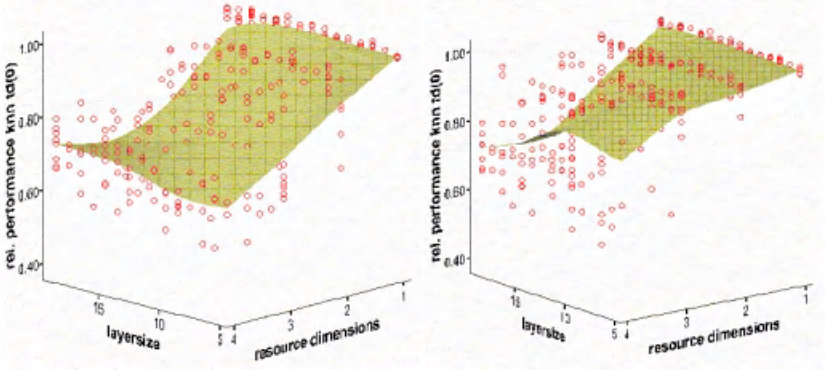


Fig. 9. Relative performance of $TD(0)$ Reinforcement Learning based on a 3-Layer-ANN value function (left: 50.000 evaluations, right: 500.000 evaluations)

Our implementation of the JAVA modules for *MDPs* and *ANNs* had to be adapted to represent the *RL* version of the decision process. *ANNs* are still necessary, since *RL* does not solve the problem of representing high-dimensional state spaces. Nevertheless, an important advantage results from the fact, that

ANNs will now be trainable by back-propagation: although, in the beginning, the training samples will be as wrong as the value function represented by the *ANN*, this only delays the training but does not prevent convergence, because *RL* can be fed back to the *ANN* after every single decision.

4.6 Results of Reinforcement Learning Compared to Genetic Algorithms

For our test problems with 1 to 4 resource dimensions we may, once again, directly compare the performance of our $TD(0)$ approach to the optimal *SDP* results, since all states on all stages can still be represented directly.

By using our random problem generator, we get the results shown in Fig. 9: after 20.000 evaluations a quality less than 5% from the optimum is reached on average and no significant negative correlation with the number of resource dimensions is observed. For a value function represented by an *ANN*, however, 50.000 evaluations are still insufficient for an acceptable approximation of the value function. Even with 500.000 evaluations training seems to have finished only for networks with small hidden layers. However, by using a higher learning rate α this convergence can be accelerated. If we compare these results (Fig. 9) to the relative performance of the *GA+ANN* approach after 500.000 evaluations (Fig. 7), the strong superiority of the $TD(0)$ approach gets evident.

5 Summary and Outlook

In summary we illustrated that dynamic pricing of (automated) information services for an anonymous market demand shows high structural similarities to *YM* problems known from other service industries. Although minor differences exist, adapting yield management seems to offer a more promising road than adapting classical price theory.

When comparing *RL* with *GAs* „breeding“ good decision functions or classical approaches of pricing in *SDPs*, *RL* clearly turned out to be the superior choice, although our results show that for a higher number of resources dimensions, we still have to use an appropriate “compression” of the state space and it is not at all clear, that the *Multi-Layer Perceptron* (*MLP*) we used is the best choice. We will therefore explore the following alternatives:

- *Vector Quantization Methods* ([20,21]), using a much finer granularity of the state space in those regions that are highly „used“ at the expense of a coarse grain representation in other regions
- *Kohonen Maps* ([17,18,19,29]), making only those neurons interact, which exhibit a certain proximity relation (compared to the *MLP* relating all neurons of one layer to all neurons of the next one)
- *Neural Gas-Algorithms* ([24,23]), also restricting learning to the local neighborhood but using an Euclidean-metric to determine the strength of interaction rather than the fixed neighborhood structure of *Kohonen Maps*.

Up to now, only supply-side complementarities of goods/services bundles for a single customer have been addressed. Integrating demand-side externalities between customers, known to be particularly strong for *IPIS* would be a promising extension, while employing *RL* methods.

Another perspective can be seen in a game theoretic context, when the resource allocation is done by a community of *RL* agents, each of trying to maximize its own profit by applying *YM*.

References

1. Alstrup, J., Boas, S., Madsen, O. et al.: Booking Policy for Flights with two Types of Passengers. European Journal of Operational Research 27 (1986) 274-288
2. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (1957)
3. Belobaba, P.: Application of a Probabilistic Decision Model to Airline Seat Inventory-Control. Operations Research 37 (1989) 183-197
4. Bertsch, L.: Expertengestützte Dienstleistungskostenrechnung. Poeschel, Stuttgart (1990)
5. Bertsekas, D.; Tsitsiklis, J.: Neuro-Dynamic Programming. Athena Scientific, Belmont MA (1996)
6. Bertsimas, D.; Popescu, I.: Revenue Management in a Dynamic Network Environment, Submitted to Transportation Science (2000)
7. Bishop, C.: Neural Networks for Pattern Recognition. Oxford University Press, Oxford (1995)
8. Chen, V. C., Günther, D., Johnson, E.: Airline Yield Management: Optimal Bid Prices for Single Hub Problems without Cancellations, submitted to Journal of Transportation Science (1999)
9. Debreu, G.: Theory of Value. John Wiley and Sons, New York (1959)
10. Doerninger, W.: Approximating General Markovian Decision Problems by Clustering their State- and Action-Spaces; Math. Operationsforschung und Statistik; Ser. Optimization 15, (1984) 135-144
11. Goldberg, D. E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Reading MA (1989)
12. Hildenbrand, W., Kirman A. P.: Introduction to Equilibrium Analysis, Amsterdam (North Holland) (1976)
13. Holland, J. H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and AI. University of Michigan Press, Ann Arbor MI (1975)
14. Hornick, S.: Value Based Revenue Management - A New Paradigm for Airline Seat Inventory Control. In Behrendt, R.; Bertsch, L. H., (ed.) Advanced Software Technology in Air Transport 91, AIT-Verlag, Halbergmoos (1991)
15. Humair, S.: Yield Management for Telecommunication Networks: Defining a New Landscape. Dissertation, Massachusetts Institute of Technology, Cambridge MA (2001)
16. Kimes, S.: Yield Management. A Tool for Capacity Constrained Firms. Journal of Operations Management 4 (1989) 348-363
17. Kohonen, T.: Self-organized Formation of Topologically Correct Feature Maps. In: Biological Cybernetics 43 (1982) 59-69
18. Kohonen, T.: Analysis of a Simple Self-Organizing Process. In: Biological Cybernetics 44 (1982) 135-140

19. Kohonen, T.: Self-Organizing Maps. Springer Series in Information Sciences, Springer, Berlin (1995)
20. Lloyd S.: Least squares quantization in PCM. IEEE Trans. Inform. Theory, vol. IT-28, (1982) 2
21. MacQueen, J.: Some methods for classification an analysis of multivariate observations. In: LeCam, L.; Neyman, J. (eds.): Proc. of the 5^{th} Berkeley Symposium on Mathematical Statistics, and Probability 1, University of California Press, Berkeley CA (1967) 281-297
22. Malinvaud, E.: Lectures on Microeconomic Theory. Amsterdam (North Holland), 4^{th} ed. (1974)
23. Martinez, T. M.; Schulten, K. S.: A "Neural-Gas" Network Learns Topologies. In: Kohonen, Teuvo.; Mäkisara, K.; Simula, O.; Kangas, J. (eds.): Artificial Neural Networks, North Holland, Amsterdam (1991) 397-402
24. Martinez, T. M.: Competitive Hebbian Learning Rule Forms Perfectly Topology Preserving Maps. In: ICANN '93: International Conference on Neural Networks, Springer, Amsterdam (1993) 427-434
25. Nieschlag, R., Dichtel E., Hörschgen H.: Marketing. 17. Auflage, Duncker & Humboldt, Berlin (1994)
26. Nollau V., Hahnewald-Busch A.: An Approximation Procedure for Stochastic Dynamic Programming in Countable State Spaces; Math. Operationsforschung und Statistik; Ser. Opt. 9 (1978) 109-117
27. Puterman, M.: Markov Decision Problems. Wiley, New York (1994)
28. Remmers, J.: Yield Management im Tourismus. In: Schertler (ed.) Tourismus als Informationsgeschäft. Ueberreuter, Wien (1994)
29. Ritter, H.; Martinetz, T.; Schulten, K.: Neuronale Netze: Eine Einführung in die Neuroinformatik selbst organisierender Netzwerke. Addison-Wesley, Reading MA (1990)
30. Schwefel, Hans-Paul: Evolution and Optimum Seeking. Wiley-Interscience, New York (1995)
31. Simon, H.: Preismanagement. 2. Auflage, Gabler, Wiesbaden (1992)
32. Smith, B., Leimkuhler, J., Darrow, R.: Yield Management at American Airlines. Interfaces 1 (1992) 8-31
33. Sutton, R.; Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge MA (1999) 52
34. Talluri, K., Ryzin, G.: An Analysis of Bid-Price Controls for Network Revenue Management. Working Paper of the Management Science and Operations Management Division, Columbia Business School, Columbia University New York (1996)
35. Vogel, H.: Yield Management - Optimale Kapazität für jedes Marktsegment zum richtigen Preis. Fremdenverkehrswirtschaft International 22 (1998)
36. Wetherford, L., Bodily S.: A Taxonomy and Research Overview of Perishable-Asset Revenue Management: Yield Management, Overbooking and Pricing. Operations Research 40 (1992) 831-844

Incremental Fuzzy Decision Trees

Marina Guetova, Steffen Hölldobler, and Hans-Peter Störr*

Artificial Intelligence Institute
Department of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
{sh,hans-peter}@inf.tu-dresden.de

Abstract. We present a new classification algorithm that combines three properties: It generates decision trees, which proved a valuable and intelligible tool for classification and generalization of data; it utilizes fuzzy logic, that provides for a fine grained description of classified items adequate for human reasoning; and it is incremental, allowing rapid alternation of classification and learning of new data. The algorithm generalizes known non-incremental algorithms for top down induction of fuzzy decision trees, as well as known incremental algorithms for induction of decision trees in classical logic. The algorithm is shown to be terminating and to yield results equivalent to the non-incremental version.

Keywords: incremental learning, classification, decision trees, fuzzy logic

1 Introduction

Decision trees have proven to be a valuable tool for description, classification and generalization of data. This is related to the compact and intelligible representation of the learned classification function, and the availability of a large number of efficient algorithms for their automated construction [8]. They provide a hierarchical way to represent rules underlying data. Today, a wealth of algorithms for the automatic construction of decision trees can be traced back to the ancestors ID3 [9] or CART [3].

In many practical applications, the data used are inherently of imprecise and subjective nature. A popular approach to capture this vagueness of information is the use of fuzzy logic, going back to Zadeh [12]. The basic idea of fuzzy logic is to replace the "crisp" truth values **1** and **0** by a degree of truth in the interval $[0, 1]$. In many respects, one can view classical logic as a special case of fuzzy logic, providing a more fine grained representation for imprecise human judgments. Consider, for instance, the question whether a jacket is fashionable. Here, a simple yes/no judgment looses information, or might even be infeasible.

To combine the advantages of decision trees and fuzzy logic, the concept of a decision tree has been generalized to fuzzy logic, and there is a number of algorithms creating such trees, e.g. [7,13]. While there are alternatives, we feel the fuzzy decision trees have some merits not easily achievable by other methods. One approach to eliminate unfortunate

* This author acknowledges support by hybris GmbH, Munich, within the project "Intelligent Systems in e-Commerce" (ISeC).

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 67–81, 2002.

need to have only yes/no judgments is the use of continuous attribute values, as done in many algorithms, e.g. Quinlans C4.5 [10]. However, there is a difference what continuous attributes and fuzzy values can naturally express. While decision trees with continuous attributes can capture more fine grained decisions (e.g. "if the attribute temperature is between 25.3 and 27.7 the state is classified as safe"), exactly this expressiveness can turn into a problem, since the algorithm has to determine the split points itself. Fuzzy decision trees do not admit such fine grained distinctions of the fuzzy truth values (which would not be appropriate in our application, anyway), and thus do not have to make the sometimes computational expensive decision between the many possible split points of one continuous attribute. On the other hand, fuzzy logic is able to generalize multi-valued attributes naturally: the value for one attribute, described by a linguistic variable in fuzzy logic, can be chosen freely out of a multi-dimensional continuum of truth-degrees for all linguistic terms (corresponding to the attributes values) of that linguistic variable. Furthermore, unlike the approaches based on classical logic, a fuzzy decision tree is able to capture degrees of truth in the calculated classification for new examples; "crisp" truth values yielded only after the optional defuzzification.[1]

A complimentary set of approaches is the use of decision trees for probability estimation, as e.g. in CART [3], which can also be viewed as a generalization of the classical logic. We do not have an easy answer to the relation of those to approaches using fuzzy logic; yet we feel in our case fuzzy logic captures better the nature of human intuition: in our example, a jacket cannot be described as fashionable with 30% probability.

To the best of our knowledge, the present algorithms for induction of fuzzy decision trees are non-incremental. However, in many applications the collection of new data and the application of the learned function is intertwined. For instance in our project "Intelligent Systems in e-Commerce" (ISeC) [2] we try to approximate the preferences of the customers using an e-shop online. Each new web page visited by a user is customized to the learned preferences of the user, and each visit of a web page yields new data on the user behavior, which are used to update the preference function memorized for the user. In such an application non-incremental classification algorithms meet their limits, because they have no way to adapt their results to the new data, but rather have to be restarted from scratch. In such an application, incremental algorithms almost always perform better, because they simply adjust their results.

In other words, we are looking for classification algorithms that are (i) based on decision trees, (ii) utilize fuzzy logic, and (iii) are incremental. Surprisingly, there are non-incremental algorithms for fuzzy decision trees [7,13], and incremental algorithms for decision trees [11], but we have found no algorithm that satisfies all three properties. The work presented in this paper is an attempt to fill this gap.

This paper discusses the general ideas underlying our algorithm for incremental induction of fuzzy decision trees (IIFDT). In [5] the algorithms are presented in more detail. Moreover it contains the proofs which are omitted for space reasons.

[1] An interesting approach that has been suggested, is to to break examples into fractional examples (as used in C4.5 to deal with unknown attributes) capturing the different attribute values. Under some circumstances this can lead to similar results as the fuzzy decision trees, but the number of fractional examples needed is exponential in the number of attributes an example is split on.

After introducing our notation in Sect. 2, we will shortly present the concept of decision trees based on fuzzy logic (Sect. 3). Starting from the general idea of top-down induction of such trees (Sect. 4, based on Janikows FID [7]), we move in Sect. 5 and 6 to an incremental algorithm along a path similarly to Utgoff et. al.'s algorithm for incremental tree induction in the classical case [11]. Section 7 states our results on termination of the incremental algorithm and on the identity of the decision trees generated by the incremental and non-incremental algorithm. We conclude with a discussion in Sect. 8.

2 Notations

Let us first introduce some basic notions. A **fuzzy set** μ over a set $\mathcal{X}$ is a mapping from $\mathcal{X}$ into the interval $[0, 1]$ of fuzzy degrees of truth. In this paper, fuzzy sets are denoted by small Greek letters. The **size** of a fuzzy set μ over $\mathcal{X}$ is defined as $|\mu| = \sum_{x \in \mathcal{X}} \mu(x)$. Let $\mathcal{U}$ be a universe of examples. A **condition** is a fuzzy set over $\mathcal{U}$. This generalizes the concept of attribute–value pairs often used for describing the examples: to each attribute–value pair (a, v) corresponds a fuzzy set $\mu_{a,v}$ that specifies the degree $\mu_{a,v}(e)$ an example e is described by the attribute–value pair (a, v). In the terminology of fuzzy logic, an attribute corresponds to a linguistic variable, and a value to a linguistic term. But we do not use these notions here explicitly. Because we use only binary tests in this paper, we write, e.g., $\mu_{technical}$ and $\overline{\mu_{technical}}$ instead of $\mu_{technical,true}$ and $\mu_{technical,false}$.

A learning/classification task consists of a finite *set of training examples* $\mathcal{L} \subseteq \mathcal{U}$, a finite set $\mathcal{T}$ of conditions that can be used to describe examples, a finite set of classes $\mathcal{C}$, and a (partially known) mapping $\chi : \mathcal{U} \to \mathcal{C}$ called *classification function*, that defines for each example $e \in \mathcal{L}$ a class $\chi(e)$. The goal of a learning algorithm is to generate an internal representation describing the set of training examples, e.g. a decision tree. This representation can be used to predict the class of new examples, i.e. examples which are not in $\mathcal{L}$ and for which c is unknown. The goal is usually to reach a good generalization, i.e. that the classes of unseen examples will be predicted with a good accuracy.

Example 1. Suppose a user is given some information about products. This information is divided into several parts like a picture e_1, a short texts e_2 describing features of the product, a text e_3 representing details etc. The user gives positive/negative feedback on her interest in that particular piece of information. For example, if an image is presented and the user enlarges it, then the feedback concerning the image is positive. A second click on the image can shrink the image again, indicating negative interest. Likewise, a text can be decollapsed from its headline / collapsed into its headline by a single mouseclick, notifying the system about the interest of the user in that text. We aim to get an internal representation of the users' preferences in order to estimate her interest before presenting a new piece of information, such that we can prefer information likely to be relevant for her in the presentation.

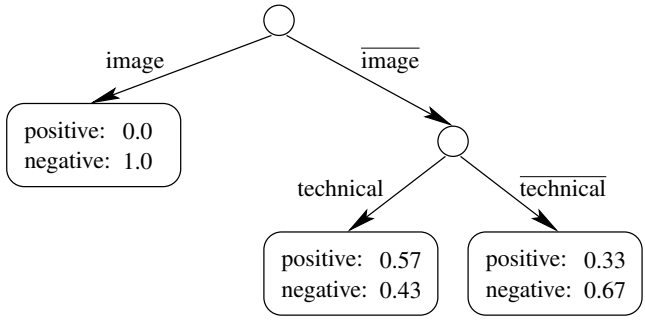


Fig. 1. A decision tree specifying the interests of the user in Example 1 considering e_1, e_2, e_3. How such a tree is calculated will become apparent later

If she is not interested in images, and weakly prefers technical information, we could have the following learning problem:

$$\mathcal{U} = \{e_1, e_2, e_3, \dots\}$$
$$\mathcal{C} = \{\textit{positive}, \textit{negative}\}$$

e	$\mu_{image}(e)$	$\mu_{technical}(e)$	$\mu_{design}(e)$	$\chi(e)$
e_1	1	0.5	0.4	*negative*
e_2	0	0.6	0.6	*negative*
e_3	0	0.8	0.5	*positive*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Finally, let us introduce some operations of fuzzy logic. Here, the operations corresponding to the classical conjunction, disjunction, and negation are the t-norm $\top$, the t-conorm $\bot$ and the complement $\bar{}$. In fuzzy logic, these functions can be chosen freely as long as they satisfy some technical conditions. As a requirement for our algorithm we restrict ourselves to the widely used so called algebraic norms

$$\top(x, y) = x * y \qquad \bot(x, y) = x + y - x * y$$

and the complement $\overline{x} = 1 - x$, where $*, +, -$ are the usual multiplication, addition and subtraction operations on real numbers.

The fuzzy operators are generalized to the operators $\cap, \cup$ and $\bar{}$ on fuzzy sets by applying the operators $\top$, $\bot$ and $\bar{}$ point-wise. Because they are commutative and associative, they can be extended to an arbitrary number of arguments. For more information about fuzzy logic the reader is referred to the widely available literature, e.g. [4].

3 Fuzzy Decision Trees

A *fuzzy decision tree* (see e.g. Fig. 1) is a tree structure where every edge is annotated with a condition, and every leaf is annotated with a fuzzy set over $\mathcal{C}$. For conciseness,

we consider only binary trees in this paper, where one of the conditions at the outgoing edges of a node is chosen from $\mathcal{T}$ (we speak of the **test condition** of the node), and the condition at the other outgoing edge is the negation of the test condition. The restriction to binary trees is lifted in [5]. A further restriction is that each condition is used at most once in each path from root to leaf.

Consider a path from the root to a leaf of the tree. Intuitively, such a path can be interpreted as follows: whenever an example fulfills the conditions the edges of a path are annotated with, we expect the example to belong to the classes according to the fuzzy set the leaf is annotated with. E.g., in Fig. 1 along the edges labelled $\overline{\text{image}}$ and technical gives us a clue, that the interest of the user in non-technical images is classified to positive with a degree of truth 0.57 and negative with a degree of truth 0.43. One should observe that unlike in decision trees based on classical logic, the conditions on the outgoing edges of a node are not necessarily mutually exclusive: e.g. an example can be both technical and non-technical to a non-zero degree of truth. So we have to combine the clues given by all paths into one final result. There is a variety of methods to accomplish this [7]. In this paper, we compute the weighted average of the fuzzy sets the leafs are annotated with, where each fuzzy set is weighted by the degree the example belongs to the conjunction of the conditions on the edges of the path. After presenting this formally, we will discuss an example. One should observe that CLASSIFY yields a fuzzy degree of truth as value; if a crisp output value is desired one can apply any of the well-known defuzzification methods [4], e.g. take the class with the highest truth value.

Example (1 continued). We classify an example

e	$\mu_{image}(e)$	$\mu_{technical}(e)$	μ_{design}
e_c	0	0.3	0.9

according to the fuzzy decision tree in Fig. 1:

Path	*leaf annotation*	*weight*	$\top$(*weight*, *annotation*)
image	$\{(pos, 0.0), (neg, 1.0)\}$	0	$\{(pos, 0.0), (neg, 0.0)\}$
$\overline{\textit{image}}$, *technical*	$\{(pos, 0.57), (neg, 0.43)\}$	$\top(1, 0.3)$	$\{(pos, 0.17), (neg, 0.13)\}$
$\overline{\textit{image}}$, $\overline{\textit{technical}}$	$\{(pos, 0.33), (neg, 0.67)\}$	$\top(1, 0.7)$	$\{(pos, 0.23), (neg, 0.47)\}$
$Classify(e_c)$			$\{(pos, 0.4), (neg, 0.6)\}$

The example is estimated negative to a truth degree 0.6. *Thus, the corresponding text would be shown in collapsed form.*

4 Induction of Fuzzy Decision Trees (FID)

Suppose we have a set of training examples $\mathcal{L}$, and we want to construct a fuzzy decision tree classifying those examples. Similarly to ID3, Janikow describes a process of top-down induction of the tree [7]. The main idea is to partition a fuzzy set of training examples according to a heuristically chosen condition recursively, until the remaining parts satisfy a pruning criterion, e.g. are either largely of one class, or of negible size. The partitioning is then represented as a tree.

$Classify(e)$:

$$Classify(e) = \sum_{\text{paths } \langle \theta_1,\ldots,\theta_i,\gamma\rangle \text{ in tree}} \top(\theta_1(e),\ldots,\theta_i(e)) \cdot \gamma \ ,$$

where a path (from root to leaf) is denoted by a vector $\langle \theta_1, \ldots, \theta_i, \gamma \rangle$ of the conditions $\theta_1, \ldots, \theta_i$ the edges of the path are annotated with and the fuzzy set γ the leaf of this path is annotated with. The scalar multiplication $\cdot$ and sum are element-wise.

Fig. 2. Algorithmus $Classify$

One should observe, that partitioning a set in the context of fuzzy logic does not necessarily mean that each example occurs in only one of the partitions to a nonzero degree. E.g. the fuzzy set $\lambda = \{(e_1, 0), (e_2, 1), (e_3, 1)\}$ partitioned by the condition $\mu_{technical}$ and $\overline{\mu_{technical}}$ yields the partitions

$$\lambda \cap \mu_{technical} = \{(e_1, 0), (e_2, 0.6), (e_3, 0.8)\}$$
$$\lambda \cap \overline{\mu_{technical}} = \{(e_1, 0), (e_2, 0.4), (e_3, 0.2)\} \ .$$

The basic idea behind the partitioning process is to reach partitions in which the vast majority of the examples belongs to only one class, i.e. the partition is "pure". Thus, if an example of unknown class belongs to such a partition, one can safely assume it is of the class the majority is in. A widely used idea to reach this goal quickly is to select a condition θ in each step such that the resulting partitions are as pure as possible. One measure for this "purity" of a partition λ is the information theoretic **information content** of the distribution of the classes in a fuzzy set λ of examples:[2]

$$IC(\lambda) = -\sum_{c \in \mathcal{C}} \left[\frac{|\lambda \cap \mu_c|}{|\lambda|} \log_2 \left(\frac{|\lambda \cap \mu_c|}{|\lambda|} \right) \right] \ ,$$

where $\mu_c = \{(e, 1) \mid e \in \mathcal{U} \ \wedge \ \chi(e) = c\} \cup \{(e, 0) \mid e \in \mathcal{U} \ \wedge \ \chi(e) \ / = c\}$.

Because the algorithm divides the set of training examples in each step in two partitions of in general different size, we can estimate the heuristic quality of the division by a condition θ as the weighted mean of the information contents of both partitions weighted by the partition size:

$$ICW(\lambda, \theta) = \frac{|\lambda \cap \theta|}{|\lambda|} IC(\lambda \cap \theta) + \frac{|\lambda \cap \overline{\theta}|}{|\lambda|} IC(\lambda \cap \overline{\theta}) \ .$$

The basic idea of the so called **information gain heuristic** is choose the condition θ for partitioning λ that has the lowest weighted information content $ICW(\lambda, \theta)$.

[2] Strictly speaking, this is not the information content, because the information content is calculated from probabilities, not from fuzzy truth degrees. However, the discussed heuristic is in close analogy to the information gain heuristic in the classical case, so this name is used here as well.

FID $(\lambda, \chi, \mathcal{T})$:

parameters: a fuzzy set of examples λ , a classification function χ, a set of conditions $\mathcal{T}$.
implicit parameters: test selection heuristic $\mathcal{H}$ and pruning criterion $\mathcal{P}$
returns a fuzzy decision tree

When the pruning criterion $\mathcal{P}$ is met, return a leaf annotated with a fuzzy set γ representing the distribution of the classes in λ:

$$\gamma = \Big\{ \sum_{e \in \mathcal{L}} \top\big(\mu_c(e), \lambda(e)\big) \;\Big|\; c \in \mathcal{C} \Big\}$$

else Choose condition θ from $\mathcal{T}$ according to a heuristic $\mathcal{H}$, and return tree:

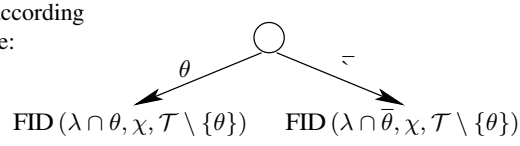


Fig. 3. Algorithm FID

Only one more ingredient is missing to formally define the algorithm FID for generation of fuzzy decision trees: we need to specify when to stop the partitioning. This is the task of a **pruning criterion**. In this paper we use a criterion that is fulfilled if the size $|\lambda|$ of the partition is below 1.5 or if more than 90% of the examples of the partition belong to one class. Both the pruning criterion and the heuristic need to be fixed during the lifetime of a tree, so we just mention these as implicit parameters in the algorithms.

In this algorithm, as well as in most of the algorithms discussed later, the restriction that in each path from root to leaf each condition occurs only once is ensured by choosing the tests from the parameter $\mathcal{T}$, that contains all conditions that may still be used in the subtree we are about to construct or modify. In the recursive calls this parameter is adjusted accordingly. The reader is invited to verify that FID called with the set of training examples $\lambda = \{(e_1, 1), (e_2, 1), (e_3, 1)\}$, the information gain heuristic, the mentioned pruning criterion, and χ and $\mathcal{T}$ chosen according to Example 1 yields the decision tree shown in Fig. 1.

The algorithm FID is by design a generalization of ID3: if all degrees of truth are 0 or 1, FID yields the same tree as ID3, and CLASSIFY returns the same classification result for new examples as the classification procedure of ID3. Note that the selection of a test condition is a computationally quite expensive process for large sets of training examples, because one has to calculate $|\lambda|$, $|\lambda \cap \mu_c|$, $|\lambda \cap \theta|$ and $|\lambda \cap \mu_c \cap \theta|$ for all $c \in \mathcal{C}$ and all conditions $\theta \in \mathcal{T}$. This will be one of the crucial points to consider in the incremental version.

5 Incremental Induction

Imagine we have a tree t constructed from a training set λ. Now we get new information and build a new tree t' that is based on a new training set λ' containing λ and a couple of new training examples. Comparing the two trees, two things may occur. First, the annotated class distributions may change, and second, the structure of the tree may change. We follow [11] that proposes to separate these changes explicitly.

In a first step, we just adapt the annotations on the tree according to the new examples without changing the structure of the tree, except turning leafs into subtrees when the pruning criterion is no longer met. Thus, the new examples are taken into considerations without computationally expensive recalculations of the tree. However, because the conditions at the inner tree nodes are not changed in this scheme, these will not necessarily be the conditions the heuristic would choose at that point if the tree was constructed with FID . So we can expect, that the tree has more nodes than necessary, and that the ability of the tree to capture the essence of the classification function will suffer. Thus, the ability to generalize the received information to classify new, unseen, examples will decrease. So we have to perform a second step once in a while, that corrects these shortcomings and ensures that the test condition at each inner node is the one the used heuristic recommends.

Our algorithm consists of four recursive procedures used to perform various tasks in updating the fuzzy decision tree. For the first step, ADDEXAMPLE performs the discussed adding of new examples without changing the structure of the tree. For the second step, OPTIMIZE checks whether structure changes in the tree are necessary, and performs the necessary changes in the tree. OPTIMIZE itself uses a procedure $Transpose$ for swapping nodes in a subtree such that the conditions on the top node of the subtree are as desired. These procedures are discussed in the next section after some preliminary considerations. Additionally, the procedure CLASSIFY (already described in Fig. 2) can be used to estimate the class of unseen examples based on the tree.

In practice, calls to ADDEXAMPLE, OPTIMIZE and CLASSIFY can be freely mixed. For testing purposes, as discussed later, we repeatedly apply the following learning regime:

1. Learn u new examples with $AddExample$.
2. Call $Optimize$ if the last optimization was at least v learned examples ago.
3. Classify w examples with $Classify$.

with the parameters, say, $u = 5, v = 50, w = 5$.

6 Restructuring the Tree

After adding new examples to the tree the split conditions are likely to be no longer **optimal** in the sense that they are the condition chosen by the given heuristic. Thus, we have (i) to find out what the best split condition is at each node, and (ii) to change the tree accordingly. As discussed before, computing the heuristic is a computationally expensive process. In the context of classical decision trees, [11] suggests to memorize the values needed to calculate the heuristics at each node of the tree, such that they need

not to be recalculated every time the condition is checked. These values are updated each time a new example is added, or the tree structure is changed. We adopt this idea for fuzzy decision trees as well. For the calculation of the information gain heuristic or a similar heuristic this concerns the values $|\lambda_t|$, $|\lambda_t \cap \mu_c|$, $|\lambda_t \cap \theta|$ and $|\lambda_t \cap \mu_c \cap \theta|$ for all $c \in \mathcal{C}$ and all conditions $\theta \in \mathcal{T}$, where λ_t is the partition of the fuzzy set λ of training examples added so far, that corresponds to each node t. I.e.:

$$\lambda_t = \lambda \cap \theta_1 \cap \cdots \cap \theta_i \ , \tag{1}$$

where $\theta_1, \ldots, \theta_i$ are the conditions the edges along the path from root to t are labelled with. To put it differently - on each node we do some bookkeeping on the set of training examples added to the subtree below that node, such that the heuristics can be calculated more efficiently.

As an addition in every node there is a *dirty* flag, that is set whenever this bookkeeping information that node is changed. As a consequence, the algorithm can skip checking the test condition of a node for optimality whenever the dirty flag is not set. Furthermore, the set λ_t needs to be memorized at each leaf node at the tree, because the leaf might be replaced by a subtree later on.

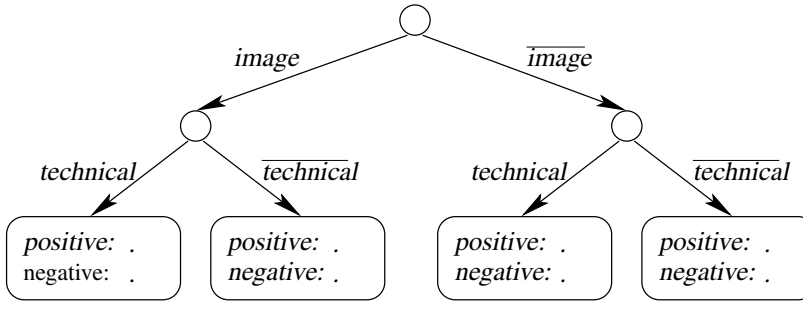
So we introduce the new concept of an **incremental fuzzy decision tree**: a fuzzy decision tree augmented with the data discussed in the previous paragraphs. For brevity, we will call these decision trees if the meaning is clear from context. A node is called **correct**, if the memorized information, that is updated by the algorithm, is equal to the values according to the definition above.

The job of the first algorithm in the suite, ADDEXAMPLE, is mostly to recursively update the memorized values at the fuzzy decision tree. If the pruning criterion is no longer met at a leaf node, the leaf node is split, so that the tree can grow to accomodate new information. In Fig. 4 the algorithm ADDEXAMPLE is given in detail.

Example (1 continued). Starting with an empty tree t, we get after adding examples e_1, e_2 and e_3 an incremental fuzzy tree like Fig. 1. (We leave out the memorized data for clarity.) Let us add a further example to the tree.

e	$\mu_{image}(e)$	$\mu_{technical}(e)$	μ_{design}	$\chi(e)$
e_4	1	0.7	0.4	*positive*

The call $AddExample(t, e_4, 1, \chi, \mathcal{T})$ modifies t to the following tree:



$AddExample(t, e, m, \chi, \mathcal{T})$:

parameters: A fuzzy decision tree with root node t, an example e and its membership value m to the tree, a classification function χ, aand set of conditions $\mathcal{T}$.
implicit parameters: test selection heuristic $\mathcal{H}$ and pruning criterion $\mathcal{P}$.
effect: Modifies t such that it takes e into account.

- If $m = 0$ return.
- If t is an inner node with edges marked θ and $\overline{\theta}$ leading to subtrees t_l and t_r then
  - update Node t with e and m.
  - $AddExample(t_r, e, \top(m, \overline{\theta(e)}), \chi, \mathcal{T} \setminus \{\theta\})$
  - $AddExample(t_l, e, \top(m, \theta(e)), \chi, \mathcal{T} \setminus \{\theta\})$
- else
  - update memorized information at Node t with e and m
  - If pruning criterion is *not* met (anymore) and $\mathcal{T} \neq \emptyset$:
    - Choose condition θ from $\mathcal{T}$ according to heuristig $\mathcal{H}$
    - Replace t by an empty tree of depth 1 with edges labelled θ and $\overline{\theta}$ and add the examples memorized on t one by one with $AddExample$.

Fig. 4. Algorithm ADDEXAMPLE

The left node was transformed to a subtree, because it did no longer fulfill the pruning criterion.

The task of TRANSPOSE is to ensure that the condition at the root node of a subtree t is the condition θ. This is done in two steps.

First, we ensure that the nodes at the 2nd level of subtree t have the test condition θ. If they are leafs, this is done by constructing an empty tree of depth 1 with the test condition θ (i.e. a tree, where all memorized values are set to 0), and adding the examples memorized at the leaf to this new subtree with ADDEXAMPLE. For the nodes at the 2nd level that are not leaves, we recursively call TRANSPOSE to ensure θ is at the top of this subtree.

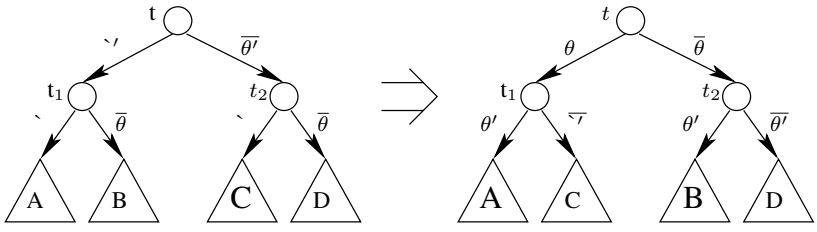
As a second step, the nodes are transposed as shown in the picture in Fig. 5, such that θ is now at the top of the tree. The crucial point is here, that we have to ensure that the information memorized on every node of the tree is correct after this whole process. In the first step, this property is ensured either by ADDEXAMPLE, or by the recursion. (More formally, in the proof of the correctness of TRANSPOSE this serves as the induction assumption.) As the reader can verify, the information memorized on the nodes of the subtrees A, B, C, D does not change because of the transposition since the set conditions on the path to the tree root does not change; neither does the memorized information at t. Unfortunately, the nodes t_1 and t_2 need further consideration, because both get new children.

Our scheme of memorizing the values needed to calculate the heuristic has the advantage that we usually do not need to go through all learned examples when restructuring a tree. We want to keep that advantage in this case as well. Here, the choice of the alge-

$Transpose(t, \theta, \mathcal{T})$:

parameters: A fuzzy decision tree t and a condition θ.
implicit parameters: test selection heuristic $\mathcal{H}$ and pruning criterion $\mathcal{P}$
returns: A fuzzy decision tree, where the edges of the top node are labelled with θ and $\overline{\theta}$.

- If t is an inner node with edges marked θ and $\overline{\theta}$ return t.
- For each child node t' of the root node of t do:
  - If t' is a leaf, replace it by an empty tree of depth 1 with edges labelled θ and $\overline{\theta}$ and add the examples memorized on t' one by one with $AddExample$,
  - else $t' := Transpose(t', \theta, \mathcal{T} \setminus \{\theta\})$.
- Swap subtrees and conditions in t as follows:



- Recalculate memorized values at t_1 resp. t_2 as sum of memorized values at A, C resp. B, D.
- If any of A, B, C and D is an empty leaf, remove it and replace its parent node by its sibling.

Fig. 5. Algorithm TRANSPOSE

braic t-norm pays off, because the memorized values at t_1 resp. t_2 are just the sum of the memorized values at the roots of subtrees A and C resp. B and D. For example:

$$\begin{aligned}
|\lambda_A| + |\lambda_C| &= |\lambda_{t_1} \cap \theta'| + |\lambda_{t_1} \cap \overline{\theta'}| \\
&= \sum_{e \in \mathcal{U}} \top(\lambda_{t_1}(e), \theta'(e)) + \sum_{e \in \mathcal{U}} \top(\lambda_{t_1}(e), 1 - \theta'(e)) \\
&= \sum_{e \in \mathcal{U}} \big[\lambda_{t_1}(e) * \big(\theta'(e) + 1 - \theta'(e)\big)\big] \\
&= |\lambda_{t_1}|
\end{aligned}$$

Summing up, a formal description of TRANSPOSE is given in Fig. 5.

Last, let us consider the last procedure, $Optimize$. Basically, it recursively traverses the tree in preorder, checks whether nodes are optimal (i.e. the test condition is the one chosen by the heuristic) and calls, if needed, TRANSPOSE to change the test conditions. Furthermore, it prunes subtrees if they fulfill the pruning condition. Note that the calculation of the heuristic in each node is computationally much less expensive than in FID because it can rely on the data memorized at the nodes of the incremental fuzzy decision tree. Figure 6 gives the algorithm in more detail.

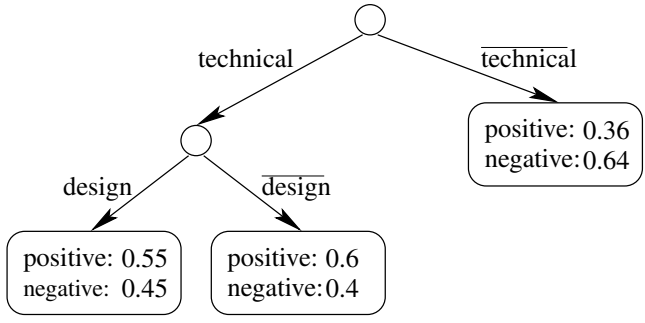
$Optimize(t, \mathcal{T})$:

parameters: A incremental fuzzy decision tree t, and a set of conditions $\mathcal{T}$.
implicit parameters: test selection heuristic $\mathcal{H}$ and pruning criterion $\mathcal{P}$.
returns: A fuzzy decision tree, where the conditions on every edge are exactly the ones chosen by the heuristic, and the pruning criteria are met at the leafs, and only at the leafs.

- If t is a leaf or $\mathcal{T} = \emptyset$ return t.
- If t meets pruning criteria $\mathcal{P}$, return leaf annotated with data calculated from t.
- If node is marked *dirty*
  - Let θ be the condition chosen from $\mathcal{T}$ for this node by the heuristic $\mathcal{H}$,
  - $t := Transpose(t, \theta, \mathcal{T} \setminus \{\theta\})$,
  - clear the *dirty* mark of the root node of t.
- Return t with all child nodes t' of the root node of t replaced by $Optimize(t', \mathcal{T} \setminus \{\theta\})$.

Fig. 6. Algorithm OPTIMIZE

Example (1 continued). The tree generated by ADDEXAMPLE for $e_1, \ldots, e_4$ is not optimal: the test heuristic recommends the test technical at the top. $Optimize(t, \mathcal{T})$ therefore modifies the tree. Due to the better choice of the test conditions, one node in the tree is pruned. Thus, the following tree represents the users interests more compactly.



7 Properties of the Algorithms

We are mainly concerned with two properties, namely termination and correctness of our algorithms. We conclude with the proof, that FID and the incremental algorithm suite yield equivalent decision trees under suitable conditions.

Theorem 1. *The algorithms* ADDEXAMPLE, TRANSPOSE *and* OPTIMIZE *terminate.*

Sketch of Proof. In the case of ADDEXAMPLE the proof is done by induction on the size of the finite set of available tests $\mathcal{T}$ in the classification task: this size decreases on every recursive call. The termination of TRANSPOSE can be proven by induction on the tree height: each recursive call is done on a tree of decreased height. The proof of

termination for OPTIMIZE is analogous to the proof of ADDEXAMPLE, but bases on the termination properties of the other algorithms. □

The next step is to verify that the algorithms produce incremental fuzzy decision trees that correctly represent the set of training examples.

Proposition 1. *The algorithms* ADDEXAMPLE, TRANSPOSE *and* OPTIMIZE *are correct.*

The proof of this proposition involves a detailed discussion of the steps of all algorithms and the calls between the algorithms. Due to lack of space we cannot present details here.

Our intention for the use of the algorithm suite is the following. Suppose a tree t is constructed by adding examples with ADDEXAMPLE to an empty tree and optimizing it with OPTIMIZE. Then t should be equivalent to the tree t' generated by FID from the same set of examples in the sense that t has the same structure and carries all the labels of t'. This ensures that $Classify$ yields the same results for both trees. Of course, t carries some more data on each node for bookkeeping reasons outlined in Sect. 6, but this does not need to concern us here, because these additional data are not used by $Classify$.

Because in most practical application of incremental algorithms one wants to interlace the classification of some examples and the addition of new training examples, we allow for optimization of the tree using $Optimize$ in between calls to ADDEXAMPLE. Formally, this tree construction procedure is defined as follows:

Definition 1. *A tree t is* ***constructed and optimized*** *from a sequence of examples $\langle e_1, e_2, \ldots e_m \rangle$ if it can be constructed by the following procedure:*

- *Initialize t with an empty leaf.*
- *For each i in $\{1, \ldots, m\}$ do:*
  - *$AddExample(t, e_i, 1, \chi, \mathcal{T})$*
  - *Nondeterministically perform $t := Optimize(t, \mathcal{T})$.*
- *$t := Optimize(t, \mathcal{T})$.*

The learning scheme introduced in Sect. 5 is an instance of this definition.

Now we can proceed to our final theorem proving the equivalence of FID and our suite of algorithms with regard to the results.

Theorem 2. *Let χ be a classification function, $\mathcal{T}$ a finite set of conditions and t a tree constructed and optimized from a sequence of examples $\langle e_1, e_2, \ldots e_m \rangle$. Let $\lambda = \{(e_i, 1) \mid i = 1, \ldots, m\}$. Then, t is equivalent to the tree constructed by* FID $(\lambda, \chi, \mathcal{T})$.

Sketch of Proof. It is easy to see that OPTIMIZE returns a tree in which every node is optimal, i.e. has the test condition chosen by the given heuristic. One can show by induction over the tree structure that for each node t we have that the subtree below t is equivalent to the result of FID $(\lambda_t, \chi, \mathcal{T}_t)$, where λ_t is defined in (1), and $\mathcal{T}_t$ is $\mathcal{T}$ reduced by the conditions that label the edges on the path from the tree root to t.

Thus, the whole tree is equivalent to FID $(\lambda, \chi, \mathcal{T})$. □

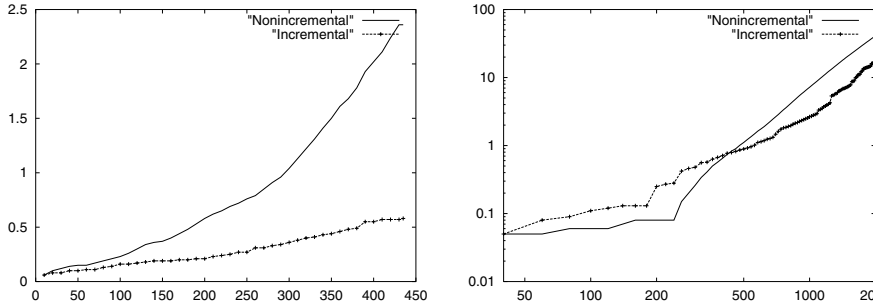


Fig. 7. Time behavior for our incremental algorithm in comparison to FID on the house votes database [1] (a), and simulated user behavior data (b). The diagram shows the computation time in seconds needed to learn the examples over the number of examples learned resp. classified

8 Discussion

In this paper we have presented a suite of algorithms for incremental induction of fuzzy decision trees. The full description and omitted proofs can be found in [5]. In fact, [5] generalizes the results presented in this paper in two aspects: Whereas we have used only binary tests in this article, non-binary tests are allowed, and methods for handling missing attributes are discussed. Furthermore, the handling of the bookkeeping data memorized at the nodes of the tree is presented in detail.

The algorithms presented here have been developed in our project to introduce personalization into a generic Web shop system. We started out from the approach of [6] to compose a web page by automatically arranged information items, like pictures, texts, or videos. These items can be collapsed / decollapsed by user actions, providing both means to adapt the web page to user preferences, as well as means to implicitly collect data about these preferences: we take the action of collapsing resp. decollapsing as an indication of negative resp. positive interest. When the user accesses a new web page, the interest of the user in the items of that page is predicted by a learning algorithm on the basis of the previous interest indications of the user and attributes the shop operator gave each item. The states of the items are then chosen accordingly.

Using a learning algorithm in classical logic, this was implemented into the hybris jakarta e-Commerce solution [2]. However, it quickly became apparent that classical logic is too inflexible for that purpose, because simple yes/no indications whether an item has a feature are often counter-intuitive. So a solution based on the fuzzy logic learning algorithm FID was implemented. To improve performance we have been developing the presented incremental algorithm suite.

In comparison to FID the incremental algorithm has the advantage that the already learned examples do not necessarily have to be reconsidered when updating the tree. On the other hand there is a considerable amount of additional bookkeeping as discussed in Sect. 6. It is not too difficult to construct an example showing that in the worst case

the incremental algorithm cannot perform better than FID . However, we argue that such worst-case examples hardly ever occur in practical applications. In practice, the incremental algorithm can save a significant amount of CPU time, especially when the learned tree more or less stabilizes its structure after learning some examples. In Fig. 7 we show a comparison of the computation times of the incremental algorithm suite in comparison to the non-incremental FID . Both algorithms were employed in a learning regime as presented at the end of of Sect. 5. This regime is meant to resemble our discussed application. Every time a new web page is shown to an user, the information about the interest of the user in the information items of the previous is added to the information memorized about the user. Then, the items of the new web page are classified by the algorithm, and arranged accordingly. When using the non-incremental algorithm, we have to recalculate the tree in this step, while it suffices in the incremental case to add the data to the tree using the relatively inexpensive $AddExample$ to update the tree, and reorganize the tree with the more expensive $Optimize$ once in a while. Figure 7 shows clear advantages of the incremental algorithms in our learning regime given a sufficient number of examples. In the future we hope to check the performance of our algorithms on data gathered from usage of a real web shop.

References

[1] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[2] Sven-Erik Bornscheuer, Yvonne McIntyre, Steffen Hölldobler, and Hans-Peter Störr. User adaptation in a Web shop system. In M. H. Hamza, editor, *Proceedings of the IASTED International Conference Internet and Multimedia Systems and Applications*, pages 208–213, Anaheim, Calgary, Zurich, 2001. ACTA Press.

[3] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth & Brooks Advanced Books and Software, Pacific Grove, CA, 1984.

[4] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, New York, 1980.

[5] Marina Guetova. Inkrementelle Fuzzy-Entscheidungsbäume. Diplomarbeit, Knowledge Representation and Reasoning Group, Department of Computer Science, Technische Universität Dresden, Dresden, Germany, January 2001, (in German).

[6] Tanja Hölldobler. *Temporäre Benutzermodellierung für multimediale Produktpräsentationen im World Wide Web*. Peter Lang Europäischer Verlag der Wissenschaften, Frankfurt, 2001, (in German).

[7] Cezary Z. Janikow. Fuzzy decision trees: Issues and methods. *IEEE Transactions on Systems, Man, and Cybernetics*, 28(1):1–14, 1998.

[8] S. K. Murthy. Automatic construction of decision trees from data: a multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2:345–389, 1998.

[9] J. R. Quinlan. Induction on decision trees. *Machine Learning*, 1:81–106, 1986.

[10] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[11] Paul E. Utgoff, Neil C. Berkman, and Jeffery A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29:5–44, 1997.

[12] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:407–428, 1965.

[13] Jens Zeidler. *Unscharfe Entscheidungsbäume*. PhD thesis, Technische Universität Chemnitz, Fakultät Informatik, 1999.

Learning from Multiple Bayesian Networks for the Revision and Refinement of Expert Systems

Michael Borth

DaimlerChrysler Research and Technology RIC/AM
Postfach 2360, 89013 Ulm, F.R. Germany
Michael.Borth@DaimlerChrysler.com

Abstract. Many expert systems for diagnosis, prediction, and analysis in complex dynamic scenarios use Bayesian networks for reasoning under uncertainty. These networks often benefit from adaptations to their specific conditions by machine learning on operational data. The knowledge encoded in these adapted networks yields insights as to typical modes of operations, configurations, types of usage, etc. To utilize this knowledge for the revision and refinement of existing and future expert systems, we developed a context-sensitive machine learning process that uses a multitude of Bayesian networks as input for concept discovery. Our algorithms allow the identification of typical network fragments, their relations, and the context in which they are valid. With these results, we are able to substitute parts of existing networks that are not yet optimally adapted to their tasks and initiate a knowledge engineering process aiming at a precise network generation for future expert systems which accounts for previously unknown characteristics.

1 Introduction

Bayesian networks offer a number of well-documented advantages for the representation and processing of knowledge and uncertainty. For example, Bayesian networks can be understood and designed by humans as well as learned from data, and real-time capable algorithms for reasoning under uncertainty exist. Also, decision-support systems built using Bayesian networks yield a wide range of functionality such as decision strategies, explanations, and conflict analyses.[1] Given these benefits for the diagnosis, prediction, and analysis of the behavior of complex dynamic systems, we naturally evaluate their use in next-generation systems with these functionalities at DaimlerChrysler Research and Technology. In this paper, we look at Bayesian networks in two distinct scenarios:

- Predictive diagnosis of complex dynamic systems (e.g. components of vehicles), taking configuration, typical modes of operations, and user behavior into account.
- Probabilistic modeling of user preferences, behavior, and short- and long-term objectives for assistance systems.

[1] See, for example, [1] for an introduction to Bayesian networks, [2] for applications, and [3] for an overview of learning of Bayesian networks.

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 82-96, 2002.

In both scenarios, we have the means to generate functional Bayesian networks: for a given technical system, we generate a Bayesian network using a translation and generation process based on domain-specific rules. This process assembles predefined network fragments for system components as per the related system specifications [4]. For user preferences, behavior, and goals, we use Bayesian networks designed by experts on the basis of past experience, results from questionnaires, and experiments. We employ object-oriented modeling techniques in both scenarios, thus generating object-oriented Bayesian networks (OOBN) as described by Koller and Pfeffer in [5].

Although networks generated in these ways perform their tasks well, they benefit from additional adaptation, either by adjustment of the network parameters (probabilities) or by structure learning to discover additional dependencies. We utilize the automatically generated and processed operational data of the systems themselves as input for these adaptations. In the first scenario, for example, we adapt the prior probabilities of component failures according to age and usage statistics and use machine learning to discover dependencies between modes of operation and faults, thus improving the quality of predictive diagnosis. In the second scenario, advanced systems for user assistance and interaction utilize statistics and machine learning algorithms to adapt to user behavior and preferences, which enables them to predict the needs of the user.

In both scenarios, we generate a multitude of Bayesian networks. All the networks within such a set are similar, since they describe the same objects, and their generation follows the same process. Yet, they are not identical, as they have been adapted to individual characteristics of the objects they describe. It is essential to understand that the networks are realized for their own purposes, independent of the knowledge discovery process for which they form the input and its application, both of which are set out in this paper.

2 A New Machine Learning Scenario

To enhance the quality of existing and future DaimlerChrysler vehicles, of services for vehicle users and fleet owners, and of decision support within the corporation, we target learning from and about the adaptations described and their results:

- We can identify the cause of system failures by discovering and analyzing groups of systems that fail – or do not fail - and which are homogeneous with regard to their properties, typical modes of operation, etc.
- We can improve diagnosis systems by adapting prior probabilities of faults according to knowledge gained from the observation of similar systems.
- We can provide a user with assistance systems of higher accuracy and faster adaptation by predicting and implementing the necessary adaptations in advance.

Thus, our goal is to deduce knowledge for feedback to the applications' domain experts and for the revision and refinement of existing and future expert systems through the adaptation of operational networks and the network generation processes. To do so, we need a new methodology, since current data mining methods are not suitable for solving these learning scenarios, as the following evaluation illustrates.

2.1 Shortcomings of Standard Knowledge Discovery

Given that we whish to learn from adaptations which are based on operational data of systems for control, diagnosis or user assistance, etc. one approach for our learning tasks is to directly utilize this operational data. But since we do not have the telecommunications or storage technology needed to deal with the amount of data generated many times a second for hundreds or thousands of variables in a large number of mobile systems, we cannot simply transfer this data to a central site and realize a standard data mining process on it.

Distributed data mining (DDM), however, addresses these issues: specialized local models are transferred to a central site together with an extract from local data. Computations on models and data then allow either the generation of a combined model or the selection of the one that best fits the combined data.[2] Yet, DDM fails to solve our scenarios, mainly because its primary aim is to generate a model similar to the one standard data mining would generate on the combined data while minimizing the necessary data flow. Distribution is considered to be merely a technical issue.

As pointed out by Provost [7], the current view of DDM is too narrow in this respect. He notes that databases are distributed and that there may be good reasons against a monolithic database. This is particularly true if data may have different meanings in different contexts, as shown in [8]. When investigating causes for vehicle component failures, for example, the context is defined by such factors as the vehicle and its properties, its area and typical mode of operation, the behavior of its driver, etc. The analysis and interpretation of a piece of data outside this context may fail completely. This context sensitivity is the first key characteristic of a class of real-world distributed machine learning problems where DDM fails.

The second characteristic of this problem class is the focus of attention. Normally, one or several items at the local sites are of interest during the learning process, but not the sites themselves. For example, during a sales analysis done by a supermarket chain, a point of interest is determining which items in the range of goods are often bought together as this will help to optimize an advertising campaign. The sites (supermarkets) are of no interest. This is different for us. Only by looking at the sites (the vehicles) in comparison to each other and at their particulars will we be able to discover the knowledge needed.

Additionally, privacy issues often prohibit the transfer of the data extracts DDM relies on from the local sites to the central site.

To summarize, we face application scenarios in which multitudes of similar Bayesian networks and their local adaptations exist, and we need to learn from the knowledge represented in the adapted networks. Standard knowledge discovery methods cannot or should not be applied to our - and similar - learning tasks; yet their shortcomings define requirements for our approach: context-sensitive knowledge discovery from a large number of data-generating sites without the need to actually transfer any local data records to a central site.

---

[2] See the collection of papers edited by Kargupta and Chan [6] for an overview of DDM. We discuss the suitability of several specific DDM approaches for our learning scenarios and applications in [8] in greater detail.

2.2 Bayesian Networks as Input for Machine Learning

The use of the adapted Bayesian networks as input for a specialized machine learning process solves the issues discussed in the previous section:

- The storage and transfer of Bayesian networks pose no problem: the encoding of information within Bayesian networks is highly efficient, as enumerations of single data sets are condensed to probability distributions and the joint probability distribution of all variables is efficiently represented as decomposition.
- The aggregation of single data items into probability distributions prevents direct conclusions about individual facts and thus ensures privacy requirements.
- Bayesian networks are probabilistic graphical models that encode information about symbolic variables within local context (see below).

For a sensible use of Bayesian networks as input, we have to observe not only the meta-information about the networks but also the information encoded in the networks and the way this information is encoded.

Meta-Information about the Networks. The relevant meta-information consists of data about the adaptation process, most importantly the confidence in the adaptation, i.e. our certainty that we are looking at useful information and not at learning artifacts. Confidence is often computed by means of a test for correctness of the model on a data sub-set, but our applications do not allow this. Instead, we follow the law of large numbers and define confidence as the number of data records used for the adaptation.

Information in the Networks. The information encoded in the networks consists of knowledge about single variables (via marginal distributions), dependencies and reciprocal influences between the variables (via conditional distributions, the graph of the network, and the resulting pathways between variables), and the complete system (via the joint probability distribution defined by its decomposition).

Use of the complete information is impossible, as the computation of the joint probability table is far too expensive. The knowledge about single variables alone, i.e. the marginal distributions without the network structure, is insufficient for complex analyses, since it neglects the dependencies. Both the existence of dependencies between variables represented by the graph of a network and the related quantifications represented by prior and conditional distributions form the relevant information for our knowledge discovery scenarios. Consequently, we use the following parts of Bayesian networks as objects-of-interest (OOIs) for our methods and algorithms:

- Elementary network fragments: A variable together with its set of parent variables, quantified by the respective conditional probability distribution (unconditional if the variable has no parents). Elementary fragments form the basic building blocks of networks, as the set of all fragments of a network is sufficient to reconstruct it.
- Pathways: A directed path between variables, consisting of all nodes of the path in the network graph. Pathways are useful for dependency analyses.
- Arcs: A parent and a child variable and the edge between them. Arcs should only be used if the existence of an arc is a valid and interesting piece of information in itself, e.g. for the discovery of causality in functional networks as defined in [9].

If needed for the application scenario, we enhance the OOIs with the marginal distributions of their variables, especially if we process given evidence. This is, for

example, necessary if we need to distinguish between the instances of a target variable such as the status of a system (e.g. defect or not). However, this may not be necessary if the goal of the application scenario is simply the discovery of influencing factors for this target variable.

Structured Encoding of Information in Bayesian Networks. The structure of a Bayesian network defines the relations of the OOIs, i.e. the connections between them, and thus the way the information is encoded. For machine learning, a structured input domain is atypical, since many domains can be described in the simple language of unstructured attribute-value pairs. However, this is inadequate if instances have some natural structure and the objects described comprise components which interrelate, as Thompson and Langley note in [10]. The conversion of a structured domain into an unstructured representation may be possible and can simplify the learning task, but, as Quinlan argues in [11], it may also prevent the learning of consistent, effective generalizations.

In our domain, this argument falls short, since any information taken out of the context defined by the complete network may be unusable: a functional dependency between two variables is meaningless if its prior probability is close to null; a similarity between two networks in all but one variable is meaningless if this variable blocks all pathways between the other variables, thus resulting in a completely different flow of information, etc.

Network Fragments and Represented Knowledge. Every OOI represents a statement about some variables of the network and their interdependencies, as does any composition of OOIs. A closer look at the knowledge represented in such fragments of networks reveals their usefulness as independent pieces of information: due to their object-oriented design, which is based on the systems' physics or flows of information, the individual parts serve as autonomous information processing units that compute their response to an input according to their internal state and function. The adapted information about state and function lies at the core of our interest as it determines system behavior.

The interrelations between parts of networks and pieces of knowledge is investigated by Laskey and Mahoney in [12], where the authors describe the translation of pieces of information from a knowledge base into network fragments. We, however, work the other way around.

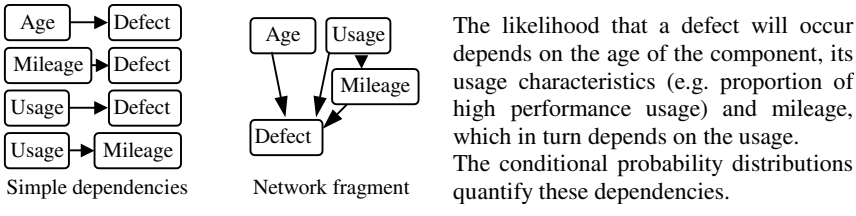


Fig. 1. Simple example of knowledge representation with network fragments with probabilities and fragment interfaces omitted for reasons of simplicity

2.3 Objectives of the Knowledge Discovery Process

To understand and implement our knowledge discovery task, we first need to define its objectives. As set out above, we work in scenarios where advanced functionalities of expert systems rely on Bayesian networks which are generated and implemented in advance and customized in a local adaptation process that works on operational data of the systems. We aim to improve both the generation and the adaptation processes. In addition, we wish to provide additional information to domain experts.

Network Generation. The reintroduction of discovered knowledge into the network generation process offers an opportunity for two improvements:

- We can replace probability distributions that are based on assumptions by distributions based on data.
- We can initiate a knowledge engineering process aiming at a finer differentiation of systems or system components according to previously unknown characteristics. For example, we can introduce several networks for a given technical system. Each network represents a prototype for the system in a typical mode of operation. Given additional characteristics, e.g. information about the configuration or a user profile, we can predict which prototype is best suited for a system and use it accordingly during network generation and configuration.

Adaptation of Networks. The same methods used for the adjustment of the network generation process can be employed for the improvement of operational networks. In this case, we utilize the adaptations carried out to identify and predict network modifications in areas that were yet not needed and were, for this reason, not adapted by local machine learning.[3]

Good examples for a potential use of this process are advanced assistance systems, which learn about the user and the user's preferences, adapt their services accordingly and additionally utilize information learned about similar users to offer further services already adapted to *predicted* preferences.

Knowledge Discovery. Knowledge discovery is important per se. For example, the identification of groups of similar networks is essential in any scenario which generates a multitude of networks. In the particular case of technical systems, the engineering departments benefit from feedback as it enables them to improve future systems. With user models, the identification of homogenous groups and their characteristics and preferences allows the development of customized services, etc.

2.4 Concept Discovery

A key aspect of all of our goals is that we need to identify groups of similar objects (systems, components, users, etc.) and find descriptions of these groups and their organization.

[3] Such an intervention requires not only bi-directional communications between the local system and the central learning site, but also strong local computational power.

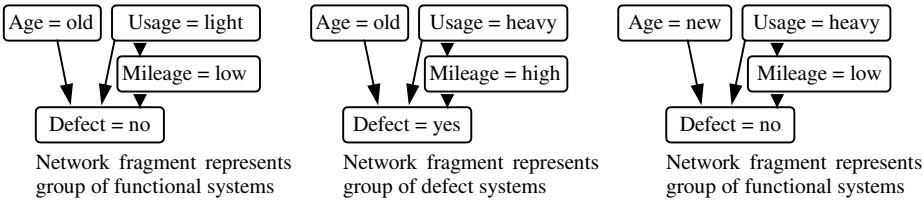


Fig. 2. Simple examples of network fragments describing typical characteristics of groups of similar systems with the given variable values representing the highest marginal probabilities

Following the definition given by Gennari, Langley, and Fisher [13], our task is to discover concepts, i.e. given a representation of objects and their descriptions, find clusterings that group these objects into concepts; find a summary description for each concept; find an organization for these concepts.

Many machine learning approaches for conceptual clustering try to find the groups of objects by maximizing a category utility function which rewards pairwise feature correlations within clusters (and sometimes additional characteristics such as simplicity) and penalizes correlations between clusters. This method maximizes intra-cluster similarity and minimizes inter-cluster similarity. On unstructured data, conceptual clustering is realized with this method using the attribute-value pairs as features and optimization strategies to find the 'best' partition of the input.

3 Concept Discovery from a Multitude of Bayesian Networks

Since we are not interested in a single optimal partition of the input, but instead in typical network fragments, the knowledge represented by them, and the context they are valid in, we have to adapt and extend the aforementioned method for conceptual clustering in several ways. Also, for each result, we require an adequate support in the input and the expression of significant knowledge.

As a framework for the resulting learning task, we developed a process called Knowledge Discovery from Models (KDM), which we first introduced in [8]. KDM is an analysis of a multitude of machine learning models aiming at parallel discovery of insights on structures within this multitude and context-sensitive information within the elements that form the structure. Thus, its objective is to discover knowledge about different pieces of information, about the contexts in which these pieces of information are valid, and about the subgroups of the input in which these contexts are given. KDM combines aspects of machine learning in structured domains and distributed data mining in order to generate the multiple results and their field of validity. While the work described below is tailored towards Bayesian networks as input, KDM itself is not restricted to a specific type of models.

The resulting knowledge discovery process is made up of these steps: we first extract the OOIs from the input networks during preprocessing (see 3.1) and then search for accumulations of them while taking the structure into account, which results in complex network fragments (see 3.2). Finally, we test these fragments for significance and category utility, which leads to a set of concepts (see 3.3) and their interrelations (see 3.4).

The focus of our analysis on network fragments to identify significant parts of the networks has the additional advantage that we do not need to change the internal representation of the information in the input for the algorithms. Moreover, network fragments can represent probabilistic combinations and aggregations of objects or features. Thus, fragments are even suitable as output of the analyses, since this allows the representation of concept hierarchies or uncertainty about network structure.

3.1 Preprocessing

During preprocessing, which is a crucial part of every knowledge discovery process, the subset of the input relevant to the task is selected and prepared. Then, the OOIs are generated together with information about where they occur:

- Network fragments, pathways, or arcs are extracted from the Bayesian networks.
- The resulting objects are tested for pairwise equality.[4] Identical objects form a group represented by a prototype (the OOI) and a list of occurrences which holds references to the input networks in which the OOI occurs.[5]

3.2 Generating Fragments

The first step of the actual concept discovery is to generate potential concepts from the OOIs and their lists of occurrences. In principle, any sensible combination of OOIs may form a concept. Since the number of these combinations increases exponentially with the number of OOIs, we are not able to examine all of them. Instead, we use a process based on the ideas of the apriori-algorithm used for the generation of association rules as introduced in [14] and analyzed in [15].

Our algorithm as depicted in figure 3 starts with the prototypes. It combines them and subsequently resulting network fragments to more complex network fragments if the support for the resulting fragment is above a minimal level provided by the user. This way, combinations which lack support are suppressed and thus not considered further in the search, resulting in the efficiency of the algorithm. The search for combinations may be implemented in several ways, e.g. as a breadth-first search that takes only joint occurrence into account and filters out results that do not consist of connected OOIs after the search. This approach works well if testing whether two OOIs form a sensible combination is computationally expensive. Alternatively, this test may be conducted during the search, which is especially efficient if a depth-first search is employed instead of the breadth-first search and the input networks vary strongly. A more intelligent search strategy is a depth-first search that looks at only the OOIs which can be combined with the current one. However, this method is only efficient for large networks because of the necessary control structures.

---

[4] We compare the similarity of the encoded information: The structure of the OOIs must be identical, while it is sufficient that the probabilities belong to the same discrete interval, defined by user-defined thresholds.

[5] Implementation details: Only prototypes are stored, not all objects that were found. Lists of occurrences need to be memory efficient and should support fast intersection operations.

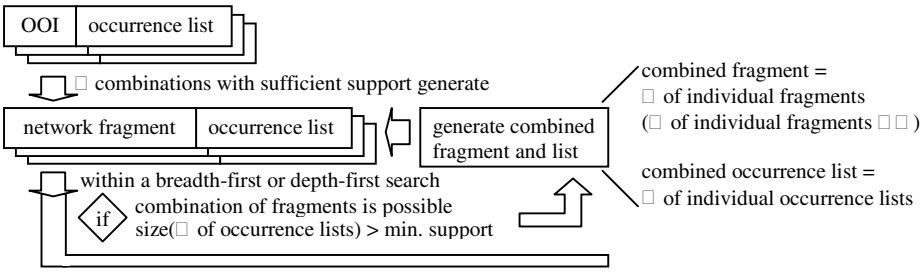


Fig. 3. Fragment generation (an overview)

In addition to having sufficient support in the input, the resulting network fragments fulfill the necessary criteria that they encode knowledge within the proper context, as the combining step of the algorithm accounts for both joint occurrence and existing interrelations.

3.3 Identifying Concepts

We now reduce the set of generated fragments to a set of concepts by first deleting redundant and superficial information. Then, from the remaining fragments we select those which offer high values for a measure of category utility.

Significance. First, we drop every fragment that is a subset of another fragment if both are supported by the same input networks. Thus, we delete every fragment that expresses no information which is not also expressed by a more specific fragment.

Second, we filter out fragments that do not encode significant knowledge. This relative measure depends on a user-defined threshold. Useful requirements for individual results to ensure minimal expressiveness are a minimal size of two or three connected edges and high information content. For the latter, we use the average entropy of the probability distributions and filter out fragments whose distributions are, on the average, close to the uniform distribution.

Category Utility. For each remaining fragment, we look at the OOIs making it up and the networks supporting it in order to determine the quality of the potential concept. This quality is measured by a category utility function *CU*, which reflects the requirements of the application scenario as well as more general characteristics. For these, Gluck and Corter introduced an objective function in [16], which rewards clusters that increase the predictability of variable values within clusters relative to their predictability in the population as a whole. In an adapted from, we use

$CU(F_k) = \alpha_k \sum_i \left[P^2(OOI_i \mid F_k) - P^2(OOI_i) \right]$ with fragment F_k, i covering all OOIs, and

α_k the average adaptation confidence in the networks supporting F_k.

If necessary, we adapt the category utility function to reflect additional characteristics tailored to the application scenarios, e.g. by rewarding fragments that decide among alternatives during dependency analyses.

Gluck and Corter's function is normally deployed as a measure of the quality of a partition of existing data as Fisher describes in [17]. We adapt the ideas behind this method to our problem class: since we wish to identify relevant pieces of knowledge, we are not interested in a single optimal partition of the input, but in all the fragments usable for a sensible partition. Hence, we compute *CU* for each fragment and drop the ones that do not fulfill user-defined criteria.

The remaining fragments express knowledge which is significant and 'typical' for a group of Bayesian networks. Therefore, these fragments fulfill the requirements of the concept discovery task of our applications.

3.4 Organization of Concepts According to Their Interrelations

Finally, we examine the interrelations of the network fragments found to understand the context in which the knowledge encoded in them is relevant. To do so, we determine which fragments are supported by which networks and how the fragments interrelate in comparison to how their sets of supporting networks relate to each other.

Using intersection operations on both network fragments and the supporting occurrence lists, we identify generic and partitive hierarchies for each result. These hierarchies describe the more general results that include the result looked at, the more specific results included in it, and the results which have a similar composition.

Though computationally expensive, the organization of the network fragments by these multi-index hierarchies improves the usability of the results in complex scenarios that result in large numbers of complex network fragments. Figure 4 illustrates a simplified example of four fragments and the hierarchies resulting from the analysis of fragment composition and occurrence lists:

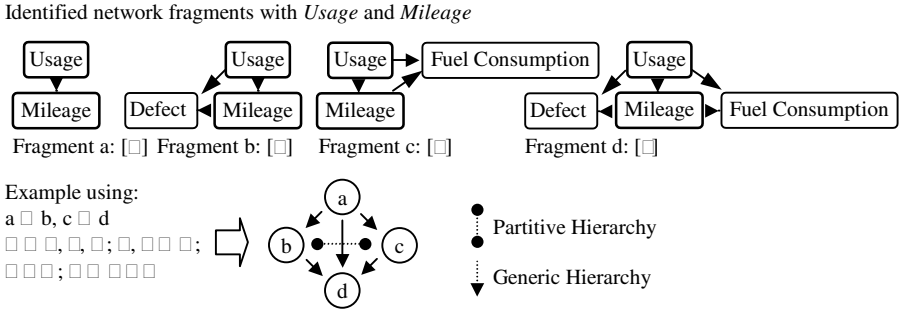


Fig. 4. Generic and partitive hierarchies for concept organization depend on the knowledge represented and supporting networks (fragments: a,b,c,d with occurrence lists □,□,□,□)

We identify a partitive hierarchy between b and c, since b □ c □ □, and □ □ □, □ □ □ □ □.

Generic hierarchies exist if a fragment x is a subset of another fragment y and the occurrence list of y is a subset of the occurrence list of x.

4 Application of the Results

4.1 Revision and Refinement of Existing and Future Networks

We utilize the generated results to reach our objectives as described in section 2.3. The most basic operation to improve Bayesian networks is the replacement of parts of the networks by more accurate parts found during the knowledge discovery process. We can make such substitutions, which are possible due to the object-oriented nature of the networks, if we do not violate the context in which the discovered fragment is relevant. This is guaranteed if we restrict the operation to networks in the same node of the respective generic hierarchy.

The analysis of different nodes in partitive hierarchies allows the refinement of the network generation process and, additionally, adaptations for existing networks. For each node, we generate networks that use the specific network fragments. A determination of characteristics of the partition, e.g. that the nodes belong to different generic hierarchies, then enables us to decide which network is particularly adapted for a given object.

4.2 Knowledge Discovery

Finally, we prepare the results for feedback to the domain experts:

Essential Characteristics. For complex fragments which represent various specific pieces of knowledge, a complete representation may feature a degree of complexity that interferes with its usability. If this is the case, we reduce this complexity by providing the essential characteristics of the concept. Essential characteristics, a standard method for the description of concepts, are defined as characteristics or descriptions an object must fulfill in order to belong to the concept (see e.g. [18]). In our learning scenario, such characteristics consist of OOIs or agglomerations of them.

We identify essential characteristics by testing if the two probabilities $P(characteristic|concept)$ and $P(\square concept|\square characteristic)$ are close to one. If so, a characteristic is considered essential.

Result Map. In addition, we compute a spatial representation of the result multitude, which provides the means to navigate the results according to personal interests. For this result map, we use the computed correlations and dependencies of the concepts: the generic and partitive hierarchies define edges in the graph of the map, whereas a measure of similarity between the concepts is useful in calculating their distance in the representation.

Even though these maps are an important part of our work, they focus on information visualization and not machine learning so that further details are outside the scope of this paper.

5 Implementation and Evaluation

5.1 Implementation

We implemented our approach to knowledge discovery from a multitude of Bayesian networks in a new software tool written in Java, called MBDiS (*M*ultitudes of *B*ayesian networks – *Di*scovery *S*ystem), a modular and expandable system, which offers methods for preprocessing, analysis, machine learning, and accessing results. It provides a graph-based human-computer interface to map the complex knowledge discovery processes.

In addition, we set up a relational database for convenient storage of and access to the Bayesian networks and network fragments. The information included in the database is sufficient for retrieval operations targeting characteristics of the networks, e.g. the fact that they contain certain variables, or the objects described by the networks. The latter is important for the application work.

5.2 Experiments

To evaluate our methods and algorithms, we experimented on artificially generated sets of networks. We generated these sets by varying several basic networks, including the well-known ALARM network [19], or networks modeling technical systems from DaimlerChrysler. The manual generation of the test sets allowed us to control the variance of the networks within the sets, i.e. their sets of variables, edges, and probability distributions.

Based on the application scenarios we look at today, we considered several distinct situations for the variation of the sets of variables and edges. For example, we expect only slight variances in the structures of networks generated in the diagnosis scenarios, but greater differences between networks that model user preferences. Table 1 shows the most important of the combinations of variances examined.

Table 1. Variances of variables and edges in the input networks

var \ edge	0%	<10%	<35%
0%			
<10%	X	X	
<35%		X	X

The percentages show the probability of non-existence for the basic network's variables and edges in the modified networks.

The resulting test sets of networks are characterized by the following figures (range of values used during experiments are set in brackets): the total number of networks (#*n* □ [70, 300]), the percentage of networks which are distinct networks (%*n* □ [15%, 80%]), the cardinality of the set of all existing variables (#*v* □ [20, 80]), the probability of non-existence for a variable in a single network (%*v* □ [0%, 50%]), the edge-to-variable ratio from the basic network (*e*/*v* □ [1.2, 2.5]), and the probability of non-existence for an edge in a single network provided that the edge exists in the basic network and parent and child variables exist in the network (%*e* □ [10%, 50%]). Based on our application scenarios, we are confident that these values cover the

variance we have to expect in real-world multitudes of Bayesian networks, with the obvious exception of the absolute numbers #*n* and #*v*.

We conducted several lines of experiments, aiming mainly at either the evaluation of the performance of the implementation or the verification of its correctness. We were also hoping to gain insight on the interrelations between the characteristics of the input multitude, the parameters used during the machine learning steps, and the results. A complete description of the experiments is outside the scope of this paper, since it focuses on the basics ideas and methods of knowledge discovery from a multitude of Bayesian networks and not on the details of the implementation. However, the following results regarding the algorithm's performance and most important parameters may help the reader to evaluate their usability.

5.3 Selected Results

Run-time performance is primarily determined by the generation of the OOIs and their lists of occurrences. Factors for the required time t are: $t \sim n$, $t \sim \#v$, $t \sim$ #OOIs (the number of OOIs, depending on $\%n$ and thus $\%v$ and $\%e$). Currently, the observed t extrapolates on average to $t < 50$ seconds for $\#n = 1000$.

The memory requirements are primarily determined by the generation of potential concept fragments. The filter effect it relies on may not be given to a sufficient extent if $\%v$ is close to 0 and $\%e < 10\%$. The exponential increase in required memory we would face without the filter effect would allow solely for small networks. Thus, our approach is restricted to application domains which either feature a minimum in variances or small networks (the actual limits depend on both the implementation and hardware used). Both of our application scenarios are such domains.

Looking at the impact of parameters, we see that the most influential parameter for the generation of the network fragments is *minimal support*, which is defined by the user. Table 2 shows the average values for *minimal support* needed for the percental reduction of the number of fragments ($\%R$) for various $\%n$:

Table 2. Impact of the minimal support parameter

$\%R$	$\%n$=80%	$\%n$=50%	$\%n$=20%
40%	5%	50%	70%
70%	10%	60%	90%
80%	15%	70%	-
90%	20%	80%	-

A linear increase of required minimal support results in increasingly smaller reductions of fragments.

Also, the filter-effect of a minimal size requirement for individual results, used to ensure a minimal expressiveness, is fairly strong: A requirement of at least two connected objects may filter out up to ~50% of the generated fragments, a requirement of three objects as much as ~70%.

For the identification of the most significant concepts, we found that the impact of the parameters varies greatly. A working policy is to use only the 25% of the results with the highest score.

6 Discussion and Directions for Future Research

Theoretical considerations, implementation, and experimental evaluation show that our approach for concept discovery from a multitude of Bayesian networks works. The generation of network fragments builds a flexible foundation for the following knowledge discovery steps and allows us to account for differences between application scenarios with regard to how the Bayesian networks encode the information relevant to the task. The context-sensitive identification of concepts and their relations enables us to discover network fragments useful for the characterization and improvement of typical networks, leading to the development of expert systems better adapted to the individual characteristics of their operation.

At this point, we plan to focus our future work on improving concept identification. For example, we plan to include user bias in the computation of the category utility. In this context, we have to consider the effort needed to gather the necessary information from the end-user. While we belief there will be a positive return on investment in our complex application scenarios, others may decide that the benefits do not overweight the costs incurred. We hope that future research will help to reduce this effort.

In addition, we need to evaluate the extend to which we can automate the revision and refinement process. A possible approach is to use autonomous agents in a distributed multi-agent architecture. The agents could use advanced pattern matching methods in order to determine whether the improved piece of knowledge they are looking at fits in with the system they are currently examining. If this is the case, they adapt the system accordingly.

7 Conclusion

The main contribution of this paper is the introduction of concept discovery from multitudes of similar Bayesian networks as they arise in our application scenarios and the utilization of these concepts for the revision and refinement of expert systems.

Our work consists of both practical and theoretical aspects: it addresses several open issues in real-world knowledge discovery scenarios where multiple data sources generate models via machine learning, the local information (and thus the knowledge encoded in the models) is highly context sensitive, and the focus of the knowledge discovery process is on information about both the single data sources themselves and items in their data. Our solution allows us to generate the information needed to provide domain experts with newly discovered knowledge, improve operational Bayesian networks, and advance the development and deployment of Bayesian networks for new systems.

Additionally, it helps us to gain insight on how Bayesian networks encode relevant information and how to represent it according to our individual requirements.

We are confident that our work will open up additional application scenarios for expert systems based on Bayesian networks and machine learning.

References

1. F. Jensen, Bayesian Networks and Decision Graphs, Springer, 2001.
2. D. Heckerman, M. Wellman: Real World Applications of Bayesian Networks, in *Communications of the ACM*, 1995.
3. D. Heckerman, A tutorial on learning with Bayesian Networks, Microsoft Research Report No MSR-TR-95-06, 1995.
4. M. Borth: Die Erstellung von Bayes-Netzen zur Diagnose Technischer Systeme, Thesis, University of Ulm, Germany, 1999 (in German).
5. D. Koller, A. Pfeffer: Object-oriented Bayesian Networks, In *Proc. of the 13th Conf. on Uncertainty in Artificial Intelligence*, 1997.
6. H. Kargupta, P. Chan (eds.): Advances in Distributed and Parallel Knowledge Discovery, AAAI Press, 2000.
7. F. Provost: Distributed Data Mining: Scaling up and Beyond. In H. Kargupta, P. Chan (eds.): *Advances in Distributed and Parallel Knowledge Discovery*, AAAI Press, 2000.
8. R. Wirth, M. Borth, J. Hipp: When Distribution is Part of the Semantics. In *Proc. of the PKDD 2001 Workshop on Ubiquitous Data Mining for Mobile and Distributed Environments*, 2001.
9. J. Pearl: Causality, Cambridge University Press, 2000.
10. K. Thompson, P. Langley: Concept Formation in Structured Domains, in *Concept Formation*, Morgan Kaufmann, 1991.
11. J. Quinlan: Learning logical definitions from relations. *Machine Learning* (5) p. 239-266, 1990.
12. K. Laskey, S. Mahoney: Network Fragments: Representing Knowledge for Construction Probabilistic Models Networks, in *Proc. of the 13th Conf. on Uncertainty in Artificial Intelligence*, 1997.
13. J. Gennari, P. Langley, D. Fisher: Models of incremental concept formation. *Artificial Intelligence* (40) p. 11-61, 1989.
14. R. Agrawal, T. Imielinski, A. Swami: Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD '93*, 1993.
15. J. Hipp, U. Güntzer, G. Nakhaeizadeh: Algorithms for Association Rule Mining - A General Survey and Comparison. *SIGKDD Explorations* 2 (1), 2000.
16. M. Gluck, J. Corter: Information, uncertainty and the utility of categories. In *Proc. of the 7th Annual Conf. of the Cognitive Science Society*, 1985.
17. D. Fisher: Iterative Optimization and Simplification of Hierarchical Clusterings, *JAIR*, vol 4 p. 147-179, 1996.
18. H. Ellis: Human Learning and Cognition, 1972
19. I. Beinlich, H. J. Suermondt, R. M. Chavez, G. F. Cooper: The ALARM monitoring system. In *Proc. of the 2nd European Conf. on Artificial Intelligence in Medicine*, 1989.

On the Problem of Computing Small Representations of Least Common Subsumers

Franz Baader and Anni-Yasmin Turhan

Theoretical Computer Science, TU Dresden, Germany
{baader,turhan}@tcs.inf.tu-dresden.de

Abstract. For Description Logics with existential restrictions, the size of the least common subsumer (lcs) of concept descriptions may grow exponentially in the size of the input descriptions. The first (negative) result presented in this paper is that it is in general not possible to express the exponentially large concept description representing the lcs in a more compact way by using an appropriate (acyclic) terminology. In practice, a second and often more severe cause of complexity was the fact that concept descriptions containing concepts defined in a terminology must first be unfolded (by replacing defined names by their definition) before the known lcs algorithms could be applied. To overcome this problem, we present a modified lcs algorithm that performs lazy unfolding, and show that this algorithm works well in practice.

1 Introduction

In an application in chemical process engineering [5,11], we support the bottom-up construction of Description Logic (DL) knowledge bases by computing most specific concepts (msc) of individuals and least common subsumers (lcs) of concepts: instead of directly defining a new concept, the knowledge engineer introduces several typical examples as individuals, which are then generalized into a concept description by using the msc and the lcs operation [1,2,9]. This description is offered to the knowledge engineer as a possible candidate for a definition of the concept.

Unfortunately, due to the nature of the algorithm for computing the lcs proposed in [2], this algorithm yields concept descriptions that do not contain defined concept names, even if the descriptions of the individuals use concepts defined in a terminology (TBox) $\mathcal{T}$. In addition, due to the inherent complexity of the lcs operation, these descriptions may be quite large (exponentially large in the size of the unfolded input descriptions). For the small DL $\mathcal{EL}$, which allows for conjunctions, existential restrictions, and the top concept, the binary lcs operation is still polynomial, but the n-ary one is already exponential. For the DL $\mathcal{ALE}$, which extends $\mathcal{EL}$ by value restrictions, primitive negation, and the bottom concept, already the binary lcs is exponential in the worst case.

To overcome the problem of large least common subsumers, we have employed rewriting of the computed concept description using the TBox $\mathcal{T}$ in order to obtain a shorter and better readable description [3]. Informally, the problem of

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 99–113, 2002.

rewriting a concept given a terminology can be stated as follows: given a TBox $\mathcal{T}$ and a concept description C that does not contain concept names defined in $\mathcal{T}$, can this description be rewritten into an equivalent smaller description D by using (some of) the names defined in $\mathcal{T}$? First results obtained in our process engineering application were quite encouraging: for a TBox with about 65 defined and 55 primitive names, source descriptions of size about 800 (obtained as results of the lcs computation) were rewritten into descriptions of size about 10 [3].

This paper complements these results in two ways. First, the positive empirical results for the rewriting approach led us to conjecture that maybe TBoxes can always be used to yield a compact representation of the lcs. More formally, our conjecture can be stated as follows. Let $\mathcal{L}$ be a DL for which the lcs operation (binary or n-ary) is exponential (like $\mathcal{EL}$ or $\mathcal{ALE}$). Given input descriptions $C_1, \ldots, C_n$ with lcs D, does there always exist a TBox $\mathcal{T}$ whose size is polynomial in the size of $C_1, \ldots, C_n$ and a defined concept name A in $\mathcal{T}$ such that $A \equiv_{\mathcal{T}} D$, i.e., the TBox defines A such that it is equivalent to the lcs D of $C_1, \ldots, C_n$? A closer look at the worst-case examples for $\mathcal{EL}$ and $\mathcal{ALE}$ from [2] supported this conjecture: the exponentially large least common subsumers constructed there can easily be represented using polynomially large TBoxes. Nevertheless, we will show in Sect. 4 that the conjecture is false, both for $\mathcal{EL}$ and $\mathcal{ALE}$.

Second, we modify the lcs algorithm presented in [2] such that it works on concept descriptions still containing names defined in a TBox. The idea is that unfolding is not performed a priori, but only if necessary. This technique, called *lazy unfolding*, is a common optimization technique for standard inferences such as subsumption [7,8], but was until now not employed for non-standard inferences like computing the lcs. Though the lcs computed by this modified algorithm may contain defined concept names, it turned out that rewriting can still reduce the size of the description. However, since it already starts with smaller descriptions, the rewriting step takes less time than with the unmodified algorithm.

2 Preliminaries

First, we introduce the DLs $\mathcal{EL}$ and $\mathcal{ALE}$ in more detail. *Concept descriptions* are inductively defined using a set of *constructors*, starting with a set N_C of *concept names* and a set N_R of *role names*. The constructors determine the expressive power of the DL. In this paper, we consider concept descriptions built from the constructors shown in Table 1. In $\mathcal{EL}$, concept descriptions are formed using the constructors top concept ($\top$), conjunction ($C \sqcap D$) and existential restriction ($\exists r.C$). The DL $\mathcal{ALE}$ provides all the constructors introduced in Table 1.

The semantics of a concept description is defined in terms of an *interpretation* $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$. The domain Δ of $\mathcal{I}$ is a non-empty set of individuals and the interpretation function $\cdot^{\mathcal{I}}$ maps each concept name $P \in N_C$ to a set $P^{\mathcal{I}} \subseteq \Delta$ and each role name $r \in N_R$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta \times \Delta$. The extension of $\cdot^{\mathcal{I}}$ to arbitrary concept descriptions is inductively defined, as shown in Table 1.

A *TBox* is a finite set of concept definitions of the form $A \doteq C$, where A is a concept name and C a concept description. In addition, we require that TBoxes

Table 1. Syntax and semantics of concept descriptions

Constructor name	Syntax	Semantics
primitive concept, $P \in N_C$	P	$P^{\mathcal{I}} \subseteq \Delta$
top-concept	$\top$	Δ
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction for $r \in N_R$	$\exists r.C$	$\{x \in \Delta \mid \exists y : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
value restriction for $r \in N_R$	$\forall r.C$	$\{x \in \Delta \mid \forall y : (x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
primitive negation, $P \in N_C$	$\neg P$	$\Delta \setminus P^{\mathcal{I}}$
bottom-concept	$\bot$	$\emptyset$

are acyclic and do not contain multiple definitions (see, e.g., [10]). Concept names occurring on the left-hand side of a definition are called *defined concepts*. All other concept names are called *primitive concepts*. In TBoxes of the DL $\mathcal{ALE}$, negation may only be applied to primitive concepts. An interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$ iff it satisfies all its concept definitions, i.e., $A^{\mathcal{I}} = C^{\mathcal{I}}$ for all definitions $A \doteq C$ in $\mathcal{T}$.

One of the most important traditional inference services provided by DL systems is computing the subsumption hierarchy. The concept description C is *subsumed* by the description D w.r.t. the TBox $\mathcal{T}$ ($C \sqsubseteq_{\mathcal{T}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{T}$. The description C is *subsumed* by D ($C \sqsubseteq D$) iff it is subsumed by D w.r.t. the empty TBox (which has all interpretations as models). The concept descriptions C and D are *equivalent* (w.r.t. $\mathcal{T}$) iff they subsume each other (w.r.t. $\mathcal{T}$). We write $C \equiv_{\mathcal{T}} D$ if C and D are *equivalent* w.r.t. $\mathcal{T}$.

In this paper, we are interested in the non-standard inference task of computing the least common subsumer of concept descriptions.

Definition 1 (Least Common Subsumer). *Let $C_1, \ldots, C_n$ be concept descriptions in a DL $\mathcal{L}$. The $\mathcal{L}$-concept description C is a* least common subsumer (lcs) *of $C_1, \ldots, C_n$ in $\mathcal{L}$ (for short $C = \mathsf{LCS}_{\mathcal{L}}(C_1, \ldots, C_n)$) iff*

1. *$C_i \sqsubseteq C$ for all $0 \leq i \leq n$, and*
2. *C is the least concept description with this property, i.e., if D is a concept description satisfying $C_i \sqsubseteq D$ for all $1 \leq i \leq n$, then $C \sqsubseteq D$.*

This definition can naturally be extended to concept definitions containing names defined in a TBox $\mathcal{T}$: simply replace subsumption by subsumption w.r.t. $\mathcal{T}$.

In general (i.e., for an arbitrary DL $\mathcal{L}$), a given collection of n concept descriptions need not have an lcs. However, if an lcs exists, then it is unique up to equivalence. This justifies to talk about *the* lcs of $C_1, \ldots, C_n$ in $\mathcal{L}$.

3 Computing the Least Common Subsumer

In [2] it was shown that, for the DLs $\mathcal{EL}$ and $\mathcal{ALE}$, the lcs always exists. The main idea underlying the algorithms computing the lcs is that concept descriptions are transformed into so-called description trees. Since subsumption can be characterized through the existence of homomorphisms between description trees, the

lcs operation corresponds to the product of description trees (see [2]). Because of space limitations, we can only outline the lcs algorithm introduced in [2]. The basic algorithm for computing the lcs of two $\mathcal{EL}$- or $\mathcal{ALE}$-concept descriptions w.r.t. a TBox consists of the following steps:

1. *Unfold the input descriptions:* if the input concept descriptions contain concept names defined in the TBox $\mathcal{T}$, these concept names are recursively replaced by their definitions until no defined names remain in the descriptions. It is well-known that the process of unfolding a description may cause an exponential blow-up [10].
2. *Normalize the input descriptions:* the normal form is computed by removing concept descriptions equivalent to $\top$, replacing inconsistent concept descriptions by $\bot$, joining value restrictions for the same role, and propagating value restrictions into existential restrictions on all role-levels. This last step of the normalization (which is only relevant for $\mathcal{ALE}$) is yet another source of an exponential blow-up [2,6].
3. *Transform the normalized descriptions into description trees and compute their product:* basically, the description tree of a normalized description is just its syntax tree. The product of the description trees can then be translated back into a concept description, which is the lcs of the input descriptions w.r.t. $\mathcal{T}$. The product construction is explained in the next subsection.

It should be noted that each of the three steps of the lcs algorithm recursively traverses the whole structure of the concept description as obtained in the previous step. The basic lcs algorithm is given as a binary operation since the n-ary lcs can be reduced to the binary operation using the fact that $\mathsf{LCS}(C_1, \ldots, C_n) = \mathsf{LCS}(C_1, \mathsf{LCS}(C_2, \ldots, C_n))$. Of course, one can also directly treat the n-ary lcs by using the n-ary product of description trees. We will illustrate the lcs algorithms for $\mathcal{EL}$ and $\mathcal{ALE}$ on two examples, which are the worst-case examples demonstrating that the n-ary lcs in $\mathcal{EL}$ and the binary lcs in $\mathcal{ALE}$ may lead to exponentially large concept descriptions (even without TBox).

3.1 The Least Common Subsumer in $\mathcal{EL}$

For the DL $\mathcal{EL}$, a *description tree* is merely a graphical representation of the syntax of the concept description. Its nodes are labeled with sets of concept names (corresponding to concept names occurring in the description) and its edges are labeled with role names (corresponding to the existential restrictions occurring in the description). We call a node w reachable from a node v by an edge labeled with r an *r-successor* of v.

For example, the trees depicted in the upper half of Fig. 1 were obtained from the concept descriptions

$$C_1^3 := \exists r.(P \sqcap \exists r.(P \sqcap Q \sqcap \exists r.(P \sqcap Q))) \ \sqcap \ \exists r.(Q \sqcap \exists r.(P \sqcap Q \sqcap \exists r.(P \sqcap Q))),$$
$$C_2^3 := \exists r.(P \sqcap Q \sqcap \exists r.(P \sqcap \exists r.(P \sqcap Q)) \sqcap \exists r.(Q \sqcap \exists r.(P \sqcap Q))),$$
$$C_3^3 := \exists r.(P \sqcap Q \sqcap \exists r.(P \sqcap Q \sqcap \exists r.P \sqcap \exists r.Q))).$$

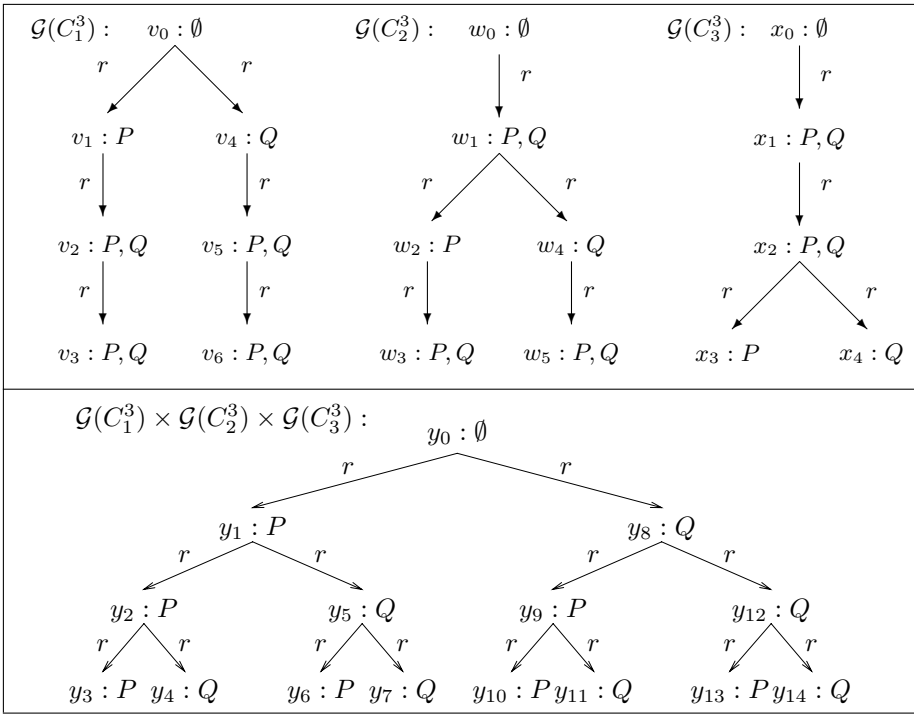


Fig. 1. Description trees of C_1^3, C_2^3, C_3^3 and their product

The *product* $\mathcal{G}_1 \times \cdots \times \mathcal{G}_n$ of n $\mathcal{EL}$-description trees $\mathcal{G}_1, \ldots, \mathcal{G}_n$ is defined by induction on the depth of the trees. Let $v_{0,1}, \ldots, v_{0,n}$ respectively be the roots of the trees $\mathcal{G}_1, \ldots, \mathcal{G}_n$ with labels $\ell_1(v_{0,1}), \ldots, \ell_n(v_{0,n})$. Then the product $\mathcal{G}_1 \times \cdots \times \mathcal{G}_n$ has the root $(v_{0,1}, \ldots, v_{0,n})$ with label $\ell_1(v_{0,1}) \cap \ldots \cap \ell_n(v_{0,n})$. For each role r and for each n-tuple $v_1, \ldots, v_n$ of r-successors of $v_{0,1}, \ldots, v_{0,n}$, the root $(v_{0,1}, \ldots, v_{0,n})$ has an r-successor $(v_1, \ldots, v_n)$, which is the root of the product of the subtrees of $\mathcal{G}_1, \ldots, \mathcal{G}_n$ with roots $v_1, \ldots, v_n$. The lower half of Fig. 1 depicts the tree obtained as the product of the description trees corresponding to the descriptions C_1^3, C_2^3, C_3^3. This tree is a full binary tree of depth 3, where the nodes reached by going to the left are labeled with P and the ones reached by going to the right are labeled with Q.

This example can be generalized to an example demonstrating that the lcs of n $\mathcal{EL}$-concept descriptions of size linear in n may be exponential in n [2].

Example 1. We define for each $n \geq 1$ a sequence $\{C_1^n, \ldots, C_n^n\}$ of $\mathcal{EL}$-concept descriptions. For $n \geq 0$ let

$$D_n := \begin{cases} \top, & n = 0 \\ \exists r.(P \sqcap Q \sqcap D_{n-1}), & n > 0 \end{cases}$$

and for $n \geq 1$ and $1 \leq i \leq n$ we define

$$C_i^n := \begin{cases} \exists r.(P \sqcap D_{n-1}) \sqcap \exists r.(Q \sqcap D_{n-1}), & i = 1 \\ \exists r.(P \sqcap Q \sqcap C_{i-1}^{n-1}), & 1 < i \leq n. \end{cases}$$

It is easy to see that each C_i^n is linear in the size of n. The product of the corresponding description trees is a full binary tree of depth n, where the nodes reached by going to the left are labeled with P and the ones reached by going to the right are labeled with Q. Obviously, the size of this tree is exponential in n. What is less obvious, but can also be shown (see [2], is that there is no smaller description tree representing the same concept (modulo equivalence).

3.2 The Least Common Subsumer in $\mathcal{ALE}$

$\mathcal{ALE}$-description trees are very similar to $\mathcal{EL}$-description trees. The value restrictions just lead to another type of edges, which are labeled by $\forall r$ instead of simply r. However, the unfolded concept descriptions must first be normalized before they can be transformed into description trees. On the one hand, there are normalization rules dealing with negation and the bottom concept. Here we will ignore them since neither negation nor bottom is used in our examples. On the other hand, there are normalization rules dealing with value restrictions and their interaction with existential restrictions:

$$\forall r.E \sqcap \forall r.F \longrightarrow \forall r.(E \sqcap F),$$
$$\forall r.E \sqcap \exists r.F \longrightarrow \forall r.E \sqcap \exists r.(E \sqcap F).$$

The first rule conjoins all value restrictions for the same role into a single value restriction. The second rule is problematic since it duplicates subterms, and thus may lead to an exponential blow-up of the description. The following is a well-known example that demonstrates this effect.

Example 2. We define the following sequence $C_1, C_2, C_3, \ldots$ of $\mathcal{ALE}$-concept descriptions:

$$C_n := \begin{cases} \exists r.P \ \sqcap \ \exists r.Q, & n = 1 \\ \exists r.P \ \sqcap \ \exists r.Q \ \sqcap \ \forall r.C_{n-1}, & n > 1. \end{cases}$$

Obviously, the size of C_n is linear in n. However, applying the second normalization rule to C_n yields a description of size exponential in n. If one ignores the value restrictions (and everything occurring below a value restriction), then the description tree corresponding to the normal form of C_n is again a full binary tree of depth n, where the nodes reached by going to the left are labeled with P and the ones reached by going to the right are labeled with Q. Figure 2 shows the $\mathcal{ALE}$-description tree of the normal form of C_3.

Given the description trees of normalized $\mathcal{ALE}$-concept descriptions, one can again obtain the lcs as the product of these trees. In this product, the bottom concept requires a special treatment, but we ignore this issue since it is irrelevant for our examples.

For each tuple of nodes on the same role-level, existential restrictions and value restrictions are treated symmetrically, i.e., for a role r the r-successors are combined with r-successors in all possible combinations (as before) and the

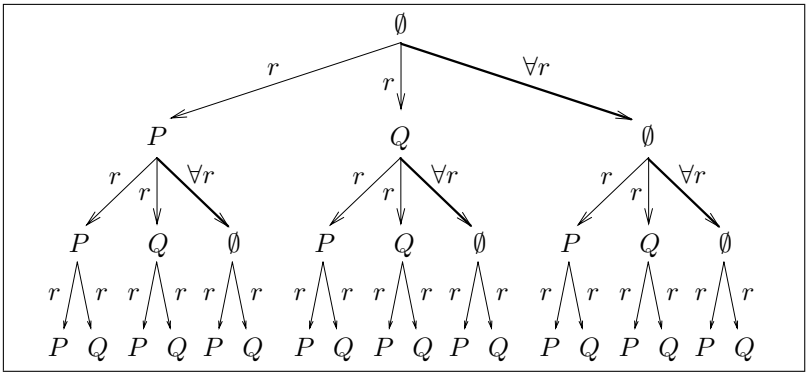


Fig. 2. The $\mathcal{ALE}$-description tree of the normal form of C_3 from Example 2

(unique) $\forall r$-successors are combined with each other. Note that r-successors are not combined with $\forall r$-successors. The following example is taken from [2].

Example 3. For $n \geq 1$, we consider the concept descriptions C_n introduced in Example 2 and the concept descriptions D_n defined in Example 1. By building the product of the description trees corresponding to the normal forms of C_n and D_n, one basically removes the value restrictions from the normal form of C_n. Thus, one ends up with an lcs E_n that agrees with the binary tree we obtained in Example 1. Again, it can be shown that there is no smaller $\mathcal{ALE}$-concept description equivalent to this lcs.

The lcs computed by the basic algorithm is a concept description not containing names defined in the underlying TBox. If some "parts" of this description have been given names in the TBox, they can be replaced by these names, thus reducing the size of the description. This can be achieved through rewriting [3]. In the next section we show that, though rewriting may be quite effective in some examples (see [3]), it cannot always reduce the size of the lcs.

4 Using TBoxes to Compress the lcs

The exponentially large lcs E_n constructed in Examples 1 and 3 has as its description tree the full binary tree of depth n, where the nodes reached by going to the left were labeled with P and the ones reached by going to the right were labeled with Q. This concept can be defined in a TBox of size linear in n.

Example 4. Consider the following TBox $\mathcal{T}_n$:

$$\{A_1 \doteq \exists r.P \sqcap \exists r.Q\} \ \cup$$
$$\{A_i \doteq \exists r.(P \sqcap A_{i-1}) \sqcap \exists r.(Q \sqcap A_{i-1}) \mid 1 < i \leq n\}.$$

It is easy to see that the size of $\mathcal{T}_n$ is linear in n and that $A_n \equiv_{\mathcal{T}_n} E_n$, i.e., the TBox $\mathcal{T}_n$ provides us with a compact representation of E_n.

In general, however, such a compact representation by structure sharing is not possible. We will first give a counterexample for the n-ary lcs in $\mathcal{EL}$, and then for the binary lcs in $\mathcal{ALE}$. The main idea underlying both counterexamples is to generate description trees having exponentially many leaves that are all labeled by sets of concept names that are incomparable w.r.t. set inclusion. To this purpose, we consider the set of concept names $N_n := \{A_j^0, A_j^1 \mid 1 \leq j \leq n\}$, and define $A^{\mathbf{i}} := A_1^{i_1} \sqcap \ldots \sqcap A_n^{i_n}$ for each n-tuple $\mathbf{i} = (i_1, \ldots, i_n) \in \{0,1\}^n$.

4.1 The Counterexample for $\mathcal{EL}$

For all $n \geq 1$ we define a sequence $C_1, \ldots, C_n$ of n $\mathcal{EL}$-concept descriptions whose size is linear in n:

$$C_j := \exists r. \bigsqcap_{B \in N_n \setminus \{A_j^0\}} B \quad \sqcap \quad \exists r. \bigsqcap_{B \in N_n \setminus \{A_j^1\}} B.$$

Since each of the concepts C_j contains two existential restrictions, the lcs of $C_1, \ldots, C_n$ contains 2^n existential restrictions. The concept descriptions occurring under these restrictions are obtained by intersecting the corresponding concept descriptions under the existential restrictions of the concept descriptions C_j. It is easy to see that these are exactly the 2^n concept descriptions $A^{\mathbf{i}}$ for $\mathbf{i} \in \{0,1\}^n$ introduced above. Since the descriptions $A^{\mathbf{i}}$ are pairwise incomparable w.r.t. subsumption, it is clear that there is no smaller $\mathcal{EL}$-concept description equivalent to this lcs. We show now that a TBox cannot be used to obtain a smaller representation.

Recall that acyclic TBoxes can be unfolded by replacing defined names by their definitions until no more defined names occur on the right-hand sides [10]. If the defined name A represents the lcs of $C_1, \ldots, C_n$ w.r.t. a TBox, then the description defining A in the unfolded TBox is equivalent to this lcs.

Obviously, to get a more compact representation of the lcs using a TBox, one needs duplication of concept names on the right-hand sides of the TBox. During unfolding of the TBox, this would, however, lead to duplication of subconcepts. Since the (description tree of the) lcs we have constructed here has 2^n different leaves, such duplication does not help, since it can only duplicate leaves with the same label, but not generate leaves with different labels. Thus, in general, we cannot represent the lcs in a more compact way by introducing new definitions in an $\mathcal{EL}$ TBox.

4.2 The Counterexample for $\mathcal{ALE}$

For $n \geq 1$ we define concept descriptions C_n of size quadratic in n. For $n \geq 1$, let $F_j^i := \forall r. \cdots \forall r. A_{j+1}^i$ be the concept description consisting of j nested value restrictions followed by the concept name A_{j+1}^i. We define

$$\begin{aligned} C_1 &:= \exists r. A_1^0 \sqcap \exists r. A_1^1, \\ C_n &:= \exists r. F_{n-1}^0 \ \sqcap \ \exists r. F_{n-1}^1 \ \sqcap \ \forall r. C_{n-1} \qquad \text{for } n > 1. \end{aligned}$$

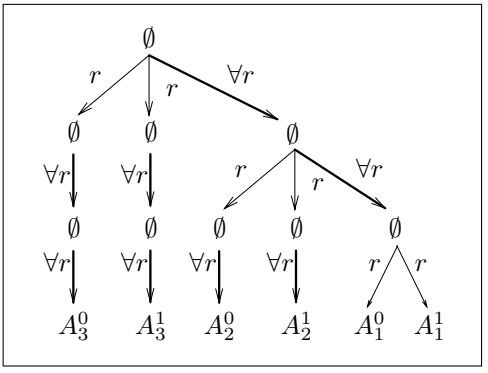


Fig. 3. The $\mathcal{ALE}$-description tree corresponding to C_3

Figure 3 shows the description tree corresponding to C_3.

Applying the normalization rule $\forall r.E \sqcap \exists r.F \longrightarrow \forall r.E \sqcap \exists r.(E \sqcap F)$ to C_n yields a normalized concept description whose size is exponential in n. If one ignores the value restrictions (and everything occurring below them), then the description tree corresponding to this normal form of C_n is a full binary tree of depth n whose 2^n leaves are labeled by the 2^n concept descriptions $A^{\mathbf{i}}$ for $\mathbf{i} \in \{0,1\}^n$.

Let $D_n := \exists r. \cdots \exists r. \bigsqcap_{B \in N_n} B$ be the concept description consisting of n nested existential restrictions followed by the conjunction of all concept names in N_n. Again, by building the product of the description trees corresponding to the normal forms of C_n and D_n, one basically removes the value restrictions from the normal form of C_n. Thus, the lcs corresponds to the full binary tree of depth n whose leaves are labeled by the concept descriptions $A^{\mathbf{i}}$ for $\mathbf{i} \in \{0,1\}^n$.

By an argument similar to the one for $\mathcal{EL}$ one can show that there is no smaller $\mathcal{ALE}$-concept description equivalent to this lcs, and that a TBox cannot be used to obtain a smaller representation.

The examples given above show that the exponential size of the lcs cannot be avoided by employing structure sharing (i.e., replacing common substructures by a defined name). In practice, however, the complexity of unfolding concept descriptions before applying the lcs algorithm appears to be more problematic than this inherent complexity of the lcs operation. How to avoid this unfolding step is the topic of the next section.

5 Computing the lcs Using Lazy Unfolding

Recall from the beginning of Sect. 3 that the computation of an lcs is realized by three consecutive traversals of the concept descriptions: unfolding, normalization, and construction of the product. The first two steps may cause an exponential blow-up of the descriptions that are in turn the input for the next step, whereas the third step is polynomial for the binary lcs operation.

Before illustrating the shortcomings of the basic lcs algorithm by an example, let us formally define the size of a concept. The *size* $|C|$ of a concept description C is increased by 1 for each occurrence of a role name or a concept name in C (with $|\top| = |\bot| = 0$).

Example 5 (naïve lcs algorithm). Given the following TBox

$$\begin{aligned}\mathcal{T} := \{\ & C_1 \doteq A_1 \sqcap \exists r.D_1, && C_2 \doteq A_2 \sqcap \exists r.D_1,\\ & D_1 \doteq (\forall s.B_1) \sqcap (\exists s.D_2) \sqcap (\exists s.D_3), &&\\ & D_2 \doteq B_2 \sqcap B_3, && D_3 \doteq B_3 \sqcap B_4\ \},\end{aligned}$$

we compute the lcs of C_1 and C_2 as sketched in Sect. 3. After the first and second step we obtain the unfolded and normalized descriptions

$$C_i' := A_i \sqcap \exists r.\big(\forall s.B_1 \sqcap \exists s.(B_1 \sqcap B_2 \sqcap B_3) \sqcap \exists s.(B_1 \sqcap B_3 \sqcap B_4)\big)$$

for $i \in \{1, 2\}$. In the third step, the algorithm first determines the concept names appearing on the top-level of the lcs—in this case none since $\{A_1\} \cap \{A_2\} = \emptyset$. Then, the algorithm makes a recursive call to compute the lcs of the descriptions occurring in the existential restriction of C_1' and C_2', respectively. This in turn leads to recursive calls for the pair of value restrictions and for all four pairs of existential restrictions for the s-successors. As output, the algorithm yields

$$\begin{aligned}\mathsf{LCS}(C_1, C_2) = \exists r.(&\forall s.B_1 \sqcap\\ &\exists s.(B_1 \sqcap B_2 \sqcap B_3) \sqcap\\ &\exists s.(B_1 \sqcap B_3) \sqcap\\ &\exists s.(B_1 \sqcap B_3) \sqcap\\ &\exists s.(B_1 \sqcap B_3 \sqcap B_4)) \quad \text{which is of size } |\mathsf{LCS}(C_1, C_2)| = 17.\end{aligned}$$

Thus, the result is computed by three recursive traversals of the input descriptions. The size of the output description results from the actually unnecessary unfolding of D_1. Even if the redundant existential restrictions are eliminated from the result, the size of the returned lcs concept description is still quite big in comparison to the equivalent description $\exists r.D_1$. Of course, rewriting the computed lcs w.r.t. $\mathcal{T}$ would also yield this result, but it would be better to avoid obtaining such an unnecessarily large intermediate result.

The idea of lazy unfolding is to replace a defined concept in a concept description only if examination of that part of the description is necessary. Lazy unfolding unfolds all defined concepts appearing on the top-level of the concept description under consideration, while defined concepts on deeper role-levels remain unchanged. Note, however, that unfolding may still be applied iteratedly if unfolding has produced another defined name on the top-level. Once this is finished, all (negated) primitive concepts, value restrictions, and existential restrictions that would occur on the top-level of the completely unfolded concept description are visible, and one can in principle continue as in the case of the naïve lcs algorithm.

The algorithm for computing the lcs with lazy unfolding, as shown in Fig. 4, is based on the following sets:

Input: Two $\mathcal{ALE}$-concept descriptions C, D and an $\mathcal{ALE}$-TBox $\mathcal{T}$

Algorithm: $\mathsf{LCS}_{\mathsf{lu}}(C, D)$

if $C \sqsubseteq_{\mathcal{T}} D$ **then** $\mathsf{LCS}_{\mathsf{lu}}(C, D) = D$
if $D \sqsubseteq_{\mathcal{T}} C$ **then** $\mathsf{LCS}_{\mathsf{lu}}(C, D) = C$
else

$C' := \mathsf{lazy\text{-}unfold}(C, \mathcal{T})$,
$\{P_1, \ldots, P_n\} := \mathsf{prim}(C')$,
for all $r \in N_R$:
$C_0 := \mathsf{val}_r(C')$,
$\{C_1, \ldots, C_n\} := \mathsf{ex}_r(C')$,
end for

$D' := \mathsf{lazy\text{-}unfold}(D, \mathcal{T})$,
$\{P_1, \ldots, P_n\} := \mathsf{prim}(D')$,
for all $r \in N_R$:
$D_0 := \mathsf{val}_r(D')$,
$\{D_1, \ldots, D_n\} := \mathsf{ex}_r(D')$
end for

$$\mathsf{LCS}_{\mathsf{lu}}(C, D) = \bigsqcap_{P \in \mathsf{prim}(C') \cap \mathsf{prim}(D')} P \quad \sqcap$$

$$\bigsqcap_{r \in N_R} (\ \forall r.\mathsf{LCS}_{\mathsf{lu}}\ (\mathsf{val}_r(C'), \mathsf{val}_r(D'))\ ) \quad \sqcap$$

$$\bigsqcap_{r \in N_R} (\ \bigsqcap_{\substack{C_i \in \mathsf{ex}_r(C') \\ D_j \in \mathsf{ex}_r(D')}} \exists r.\, \mathsf{LCS}_{\mathsf{lu}}\ (C_i \sqcap \mathsf{val}_r(C'),\ D_j \sqcap \mathsf{val}_r(D')))$$

Fig. 4. The lcs algorithm $\mathsf{LCS}_{\mathsf{lu}}$ for $\mathcal{ALE}$ using lazy unfolding

$\mathsf{prim}(C)$: denotes the set of all (negated) primitive names occurring on the top-level of C.

$\mathsf{val}_r(C)$: denotes the conjunction of the concept descriptions occurring in value restrictions on the top-level of C, where $\mathsf{val}_r(C) := \top$ if there is no value restriction.

$\mathsf{ex}_r(C)$: denotes the set $\{C_1, \ldots, C_n\}$ of concept descriptions occurring in existential restrictions of the form $\exists r.C_i$ on the top-level of C.

The $\mathsf{LCS}_{\mathsf{lu}}$ algorithm as given in Fig. 4 tests in each recursion, if the input concepts subsume each other, in this case the lcs is trivial. The algorithm uses the function $\mathsf{lazy\text{-}unfold}()$ for unfolding the top-level of the input concept descriptions w.r.t. the input $\mathcal{ALE}$-TBox $\mathcal{T}$. Next, the auxiliary sets and concept descriptions defined above are computed. The returned lcs concept description is a conjunction of three components:

1. the conjunction of all (negative) primitive concepts appearing on both the top-level of C' and D',
2. the conjunction of all value restrictions derived from recursive calls of $\mathsf{LCS}_{\mathsf{lu}}$ for each role that has a value restriction on top-level of C' and D',
3. a conjunction of the existential restrictions derived from recursive calls of $\mathsf{LCS}_{\mathsf{lu}}$ for each combination of existential restrictions where the value restrictions are propagated "on the fly".

In contrast to the three independent recursions in the naïve algorithm, the $\mathsf{LCS}_{\mathsf{lu}}$ algorithm traverses the structure of the concept descriptions recursively only once. The three steps of the basic algorithm are now interwoven on each role-level. In particular, the normalization of the concept descriptions is realized role-level-wise (i) by the definition of the description $\mathsf{val}_r(C')$ and $\mathsf{val}_r(D')$; (ii) by including the conjuncts $\mathsf{val}_r(C')$ and $\mathsf{val}_r(D')$, respectively, in the recursive calls for the existential restrictions. The normalization rules dealing with negation and the bottom concept (which we have not described above) are taken care of by the subsumption test at the beginning of the algorithm.

There are two reasons why the new lcs algorithm may avoid computations done by the naïve algorithm. First, if one of the subsumption tests at the beginning are successful, then one of the input descriptions is returned without unfolding or normalizing any of the two descriptions. This can also happen in recursive calls of the algorithm. In particular, this also means that the returned description may still contain defined names. Second, if an existential restriction for a role r has no matching restriction in the other description, then this restriction need not be processed (i.e., its concept description is not unfolded and normalized). Consider once more Example 5 to illustrate the first effect.

Example 6 (lcs using lazy unfolding). Assume we apply the new lcs algorithm to the descriptions C_1, C_2 of Example 5. In the first step, none of the two subsumption conditions hold and the algorithm calls $\mathsf{lazy\text{-}unfold}(C_1, \mathcal{T})$ and $\mathsf{lazy\text{-}unfold}(C_2, \mathcal{T})$. There is no defined concept name to replace on top-level. Then the algorithm calls $\mathsf{LCS}_{\mathsf{lu}}$ recursively for the pair of existential restrictions. Because the subsumption test is successful, this call directly yields D_1, without considering the definition of D_1. Thus the returned concept description is $\mathsf{LCS}_{\mathsf{lu}}(C_1, C_2) = \exists r.D_1$, which has size $|\mathsf{LCS}_{\mathsf{lu}}(C_1, C_2)| = 2$.

Comparing the lcs from Example 5 to the result obtained here, we see that $\mathsf{LCS}_{\mathsf{lu}}$ needs less recursive calls with less recursion depth, and furthermore returns a smaller concept description.

Using lazy unfolding is advantageous in most cases when computing a lcs w.r.t. a TBox, but there are, of course, combinations of input concept descriptions where an exponential growth of the lcs concept description is still unavoidable.

6 Implementations of the lcs

We have implemented both, the naïve and the lazy unfolding based lcs algorithm [12] in Lisp. The FaCT system [8] is used to compute subsumption. The core of both implementations is a binary lcs function wrapped by a function that successively calls the binary lcs function.

The "old lcs" is a straightforward implementation of the fundamental algorithm outlined in Sect. 3 and discussed in [2]. It also uses an implementation of the heuristic rewriting algorithm for computing small (but not always minimal) rewritings of $\mathcal{ALE}$-concept descriptions mentioned earlier (see [3]), which we use

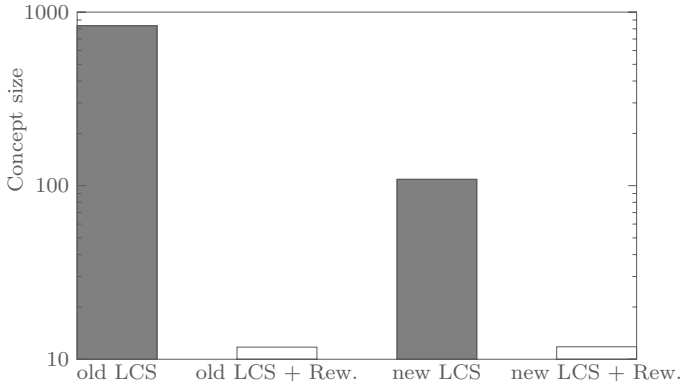


Fig. 5. Average concept sizes obtained from the four settings

in our evaluation. The "new lcs" implements the algorithm $\mathsf{LCS}_{\mathsf{lu}}$ introduced in Sect. 5. It is also a straightforward implementation of this algorithm and does not employ special low-level encoding tricks to improve its performance. In contrast to the old lcs implementation, which is strongly linked to FaCT, the new lcs may be coupled with arbitrary DL reasoners.

6.1 A First Evaluation of the Implementations

To compare the performance of our implementations of both algorithms we use a TBox derived from our application in chemical process engineering. It contains 52 primitive concepts, 67 defined concepts and 43 roles. It has a rather deep concept hierarchy, which makes it likely that least common subsumers computed for concept descriptions defined in this TBox will not collapse to $\top$.

The input descriptions we used for the evaluation are combinations of seven REACTOR concepts defined in the application TBox. To compute the lcs of all possible combinations of these REACTOR concepts, it suffices to compute some combinations determined by the attribute exploration algorithm as described in [4]. Our test suite included 22 different lcs calls, ranging from binary lcs calls to lcs calls with seven input concepts. For each computation of these least common subsumers, we measured run-times and sizes of the output concept descriptions of four settings: both of the lcs implementations and both of the lcs implementations followed by a rewriting step. The latter two use the same rewriting implementation of the heuristic algorithm. To obtain representative run-times we ran each LCS in each setting 100 times.

The results for the average concept size are shown in Fig. 5, where one should note the logarithmic scale. The measured values indicate that an lcs computed by the $\mathsf{LCS}_{\mathsf{lu}}$ implementation returns concept descriptions that are about an order of magnitude smaller than the concept descriptions returned by the naïve algorithm. The rewritten lcs concept descriptions are again one order of magnitude smaller than the lcs concept description returned by the $\mathsf{LCS}_{\mathsf{lu}}$

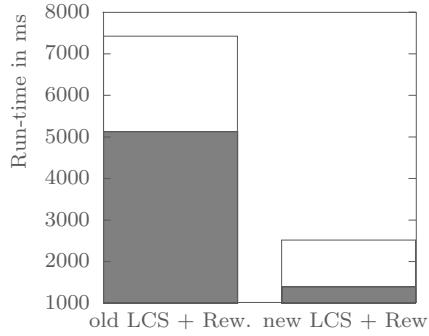


Fig. 6. Average run-times needed by the different settings

implementation. Comparing the concept sizes obtained for the settings including the heuristic rewriting shows that starting from a smaller concept description does not yield a smaller rewritten concept. This is probably due to the fact that for our examples the heuristic algorithm produced the optimal result.

The average concept description obtained with the $\mathsf{LCS}_{\mathsf{lu}}$ implementation has a concept size of about 100. These concept descriptions are still too big to be comprehensible to a human reader. In our application scenario, the knowledge engineer is supposed to choose an appropriate description from a set of computed least common subsumers, possibly also modifying the chosen description by hand. Therefore, rewriting remains necessary as an additional step in this application.

Figure 6 shows the sum of run-times for computing the lcs (grey) and rewriting the obtained lcs concept description (white). The comparison of run-times for the lcs implementations indicates a speed-up of factor 3.5. The run-time for rewriting an lcs concept description returned by the $\mathsf{LCS}_{\mathsf{lu}}$ implementation is also lower than for rewriting the description produced by the naïve algorithm (by a factor of about 2). Taking lcs computation and rewriting together, the overall run-time differs by a factor of three.

7 Conclusion and Future Work

The worst-case examples presented in Sect. 4 are quite contrived and not likely to occur in practice. Nevertheless, they show that, in principle, the exponential blow-up inherent to the lcs operation cannot be avoided, even if one can introduce "abbreviations" for subdescriptions. An interesting question for future research is to characterize situations in which this exponential blow-up cannot occur, and to check whether these situations are likely to occur in practice.

The performance of the lcs algorithm using lazy unfolding greatly depends on the structure of the TBox and the input descriptions. There are, of course, examples where there is no improvement over the naïve algorithm. However, since no overhead is generated by using lazy unfolding, it is advantageous to use the new algorithm in any case.

The first evaluation of the lcs implementations in our application framework indicates that using lazy unfolding can substantially decrease the size of the resulting concept descriptions. However, our results also indicate that it is still necessary to perform rewriting after computing the lcs in order to obtain concept descriptions that are small enough to be inspected by human users. As indicated by our tests, lazy unfolding will also decrease the run-time of the subsequent rewriting step.

References

1. F. Baader and R. Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In O. Herzog and A. Günter, eds., *Proc. of KI-98*, volume 1504 of *Lecture Notes in Computer Science*, p. 129–140, Bremen, Germany, 1998. Springer-Verlag.
2. F. Baader, R. Küsters, and R. Molitor. Computing least common subsumer in description logics with existential restrictions. In T. Dean, ed., *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99)*, p. 96–101, Stockholm, Sweden, 1999. Morgan Kaufmann, Los Altos. An extended version appeared as LTCS-Report LTCS-98-09, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 1998. See `http://www-lti.informatik.rwth-aachen.de/Forschung/Papers.html`.
3. F. Baader, R. Küsters, and R. Molitor. Rewriting concepts using terminologies. In A.G. Cohn, F. Giunchiglia, and B. Selman, eds., *Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-00)*, p. 297–308, San Francisco, CA, 2000. Morgan Kaufmann Publishers.
4. F. Baader and R. Molitor. Building and structuring description logic knowledge bases using least common subsumers and concept analysis. In B. Ganter and G. Mineau, eds., *Proc. of ICCS-00*, volume 1867 of *Lecture Notes in Artificial Intelligence*, p. 290–303. Springer-Verlag, 2000.
5. F. Baader and U. Sattler. Knowledge representation in process engineering. In *Proc. of DL-96*, 1996.
6. F. Baader and A.-Y. Turhan. TBoxes do not yield a compact representation of the least common subsumer. In *Proc. of DL-2001*, 2001.
7. F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.
8. I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.
9. R. Küsters and R. Molitor. Approximating most specific concepts in description logics with existential restrictions. In T. Eiter F. Baader, G. Brewka, eds., *Proc. of the 24th German Annual Conf. on Artificial Intelligence (KI'01)*, number 2174 in Lecture Notes In Artificial Intelligence, p. 33–47. Springer-Verlag, 2001.
10. B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence Journal*, 43:235–249, 1990.
11. U. Sattler. *Terminological knowledge representation systems in a process engineering application*. PhD thesis, RWTH Aachen, 1998.
12. A.-Y. Turhan and R. Molitor. *Using lazy unfolding for the computation of least common subsumers*. In *Proc. of DL-2001*, 2001.

Approximate Information Filtering on the Semantic Web

Heiner Stuckenschmidt

AI Department, Vrije Universiteit Amsterdam
de Boelelaan 1081a, 1081 HV Amsterdam
heiner@cs.vu.nl

Abstract. Facing the increasing amount of information available on the World Wide Web, intelligent techniques for content-based information filtering gain more and more importance. Conventional approaches using keyword- or text-based retrieval methods have been developed that perform reasonably well. However, these approaches have problems with ambiguous and imprecise information. The semantic web that aims at supplementing information sources with a formal specification of its meaning using ontologies can potentially help to overcome this problem. At the moment, however, the semantic web still suffers from its own problems in terms of heterogeneous ontologies and the need to relate them to each other. In this paper, we argue that we can overcome this problem by using shared vocabularies, a standardized language for encoding ontology that supports basic terminological reasoning (in this case DAML+OIL) and techniques from approximate reasoning. We introduce the approach on an informal level using didactic example and give a formal characterization of the method that include correctness proofs for the problem of information filtering.

1 Motivation

Today, the World Wide Web contains information about almost any topic one can imagine. This availability of information is potentially of great value for commercial as well as private users. Unfortunately, relevant information is often hidden amongst vast amounts of irrelevant information that is also available on the Web. In order to benefit from the relevant information, we have to provide sophisticated methods to separate relevant from irrelevant. This problem is also referred to as *information filtering* which is characterized as the task of removing irrelevant information from an incoming stream of unstructured textual information according to user preferences [1]. A different perspective on the same problem is that of information retrieval [11]. In information retrieval, a collection of information represented by surrogates in terms of content descriptions is searched on the basis of a user query and those documents, whose descriptions match the query are returned to the user. Many systems support the *Boolean query model* [6] that allows to state queries as Boolean expressions over keywords.

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 114–128, 2002.

Without using background knowledge on semantic correspondences between word (e.g. synonyms or hyponyms) it is very likely that the basic mechanism fails to return some relevant information. On the other hand, the presence of ambiguous word (i.e. homonyms) will disable the retrieval method to sort out all irrelevant information. As a result, the use of background knowledge has been discussed in classical information retrieval [17,7]. On the semantic web, such background information will be provided by machine processable models of information semantics that assigns information resources to classes whose meaning is defined in ontologies that serve as a unique reference for the disambiguation of concepts. The assignment to a certain ontological class provides us with a unique interpretation of the meaning of a resource thus solving the problems of keyword based approaches mentioned above. Using the concepts of a specific ontology, we can state Boolean queries over concept names with maximal precision and recall provided that all relevant information resources have been assigned to ontological categories.

While the idea of maximizing precision and recall by using concept expressions in Boolean queries is an appealing idea, the practical application on the World Wide Web suffers from the fact that there will not be 'the one' ontology that is used to classify information. We will rather face a situation, where a multitude of classification hierarchies organize different or even the same information according to different discrimination principles. A successful information filtering approach will have to make use of as many of these ontologies as possible. This claim raises the problem of comparing different ontologies. Existing approaches to this problem mainly rely on the idea of establishing mappings between ontologies or even of merging them (see contributions in [9]). We argue that for the purpose of information filtering these approaches are not appropriate, because definitions within the ontologies are likely to be changed and whole ontologies might be replaced destroying the result of the integration process.

In order to overcome the problem of heterogeneous and changing ontologies we propose an alternative approach that is based on a translation of queries that are based on concept names in order to fit the vocabulary used in different ontologies. We borrow the idea of automatically adjusting the query to an information source from [10]. While the original approach considers re-writings in order to adapt the query to comparison operators supported by the system that is queried, our approach focuses on re-writings in order to adapt to the class names that occur in the ontology serving as background knowledge for the queried information. In the following, we first describe our approach on an informal level using a didactic example. Afterwards we give a formal characterization of the filtering task, define the re-writing procedure and give and proof its correctness.

2 Overview of the Approach

In this section, we will give a general overview of our approach. It is build on two main assumptions (1) the existence of a shared vocabulary of basic terms and

(2) the existence of a common language for encoding ontologies that supports terminological reasoning. While these assumptions put hard restrictions on the applicability of the approach, they turn out to be necessary in order to provide filtering methods that are applicable across different ontologies. In the following we will first give an example of ontology heterogeneity that would disable a straightforward application of concept-based information filtering. Afterwards we will argue for the use of a shared vocabulary and explain our approach for re-writing queries.

2.1 Ontology Heterogeneity

Ontologies can differ in many ways [16]. In this paper we will not try to discuss them in general. We will rather give an example of ontologies that even though they describe the same domain of interest represent very different conceptualizations of that domain. We start with a simple ontology that discriminates animals into domestic, foreign and production animals and contains some kinds of animals that fall under one or more of these categories (compare Fig. 1).

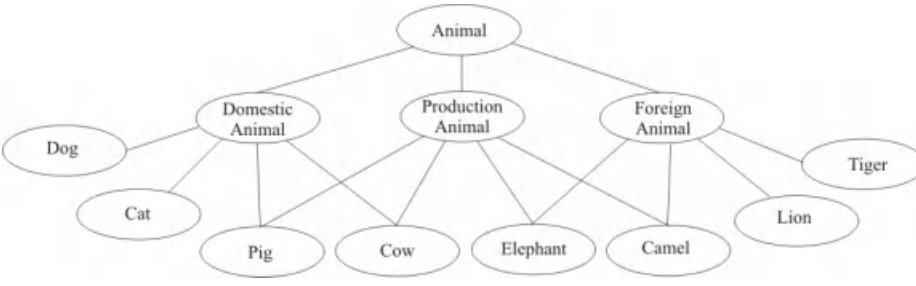


Fig. 1. An ontology of animals

Now consider an ontology that describes classes of animals in the way a child would possibly categorize them (compare Fig. 2). The main distinctions made in this ontology are pets, farm animals and zoo animals. These distinction are based on the experience of a child that some animals are kept at home, at farms or in zoos.

While both ontologies do not share any class except for the general class animal, it should be possible to establish a relation between the two. Using common world knowledge and the informal descriptions of the classes in Fig. 2 we can conclude that pets should be a subclass of domestic animal and include Cats and Dogs. Farm animals should be a subclass of domestic animals and include Cows and Pigs. Finally zoo animals should be subsumed by Foreign animals and contain all the subclasses of foreign animal shown in Fig. 2

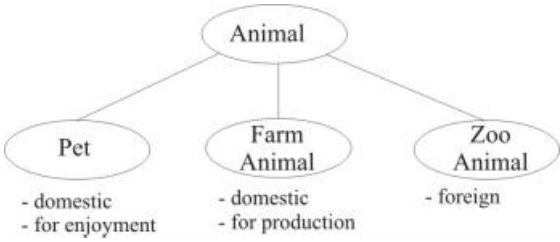


Fig. 2. A 'childish' ontology of animals

2.2 Shared vs. Non-shared Ontologies

There are different options for relating ontologies like the ones described above in order to use them for information filtering. The ones most often discussed ones are merging and mapping of ontologies [9]. In this paper, we argue for an approach that is based on on-demand translations rather than a fixed connection of ontologies. The ideas of these approaches are depicted in Fig. 3.

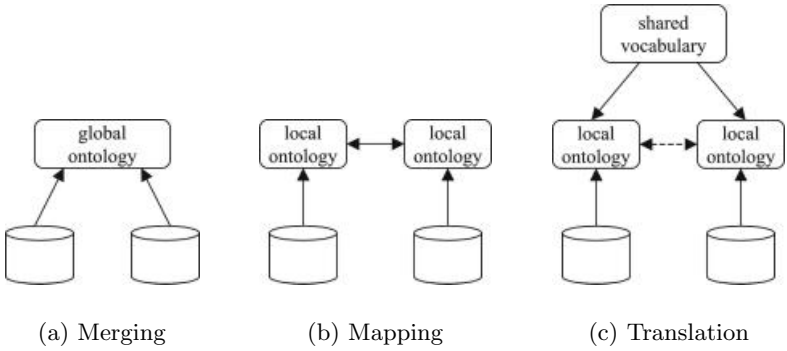


Fig. 3. Three approaches to relating ontologies

The merging approach aims at producing a single global ontology based on the ontologies to be related. The integrated ontology will contain class definitions from both ontologies arranged in a single taxonomy. The advantage of this approach is that the merged ontology contains all information that is needed to filter information and the filtering process can be implemented in a straightforward way. The drawback of this approach, however, is the effort needed to come up with a merged ontology that correctly resembles both ontologies. The situation becomes even worse if we want to include further ontologies. In this case we

have to perform further merges that are costly and lead to huge ontology that is hard to handle and maintain.

The mapping approach tries to avoid the creation of large global ontologies. It rather aims at relating concepts from both ontologies using additional knowledge, i.e. mapping rules. These rules can be used to find concepts in the other ontology that correspond to concepts used in a query and retrieve the instances of these concepts as well. The advantage of the mapping approach is that an extension to other ontology does not affect the ones already included in the system. A problem of the mapping approach is the need to specify mappings to all other ontologies already present in a system. Further, static mapping rules can be come very complicated if not only 1:1 mappings are needed like in our example.

2.3 A Translation Architecture

We propose a translation approach that is designed to combine the advantages of the mapping and the merging approach in the sense that it keeps local ontologies but connects them to a single shared ontology using special mappings: the shared ontology contains with a vocabulary that can use in order to specifies the specific semantics of information resources. This semantics, however, has to be defined individually for different information sources in terms of local ontologies, defining the meaning of specific classifications.

Our notion of a shared ontology does not require a very complex language as it mainly defines names of concepts, relations and individuals that can be used in the local ontologies as well as their relation to each other. In fact these requirements largely correspond to the expressiveness of RDF schema which we use to encode our shared ontologies. In our example, the shared ontology would for example specify the general concept animal and a set of properties and values that can be used to describe special types of animals:

```
...
<rdfs:class rdf:ID="animal"/>
<rdfs:class rdf:ID="location"/>
<rdfs:Property rdf:ID="origin">
    <rdfs:domain rdf:resource="#animal"/>
    <rdfs:range rdf:resource="#location"/>
</rdfs:property>
<location rdf:ID="domestic">
<location rdf:ID="foreign">
...
```

The part of the shared ontology given above enables us to talk about the origin of a specific animal, however, if we really want to define for example the class of animals that are of a domestic origin, we need a more expressive language. Such a language is provided by DAML+OIL [15] a proposal for a standard ontology language. Using the definitional abilities of DAML+OIL, we can easily define the classes from our example ontology using elements from the shared vocabulary. We just give the corresponding definitions of farm animals as an example:

```
<daml:Class rdf:ID="farm_animal">
    <rdfs:subClassOf rdf:resource="shared:animal"/>
    <rdfs:subClassOf>
        <daml:restriction>
            <daml:onProperty rdf:resource="shared:origin"/>
            <daml:hasValue rdf:resource="shared:domestic"/>
        </daml:restriction>
        <daml:restriction>
            <daml:onProperty rdf:resource="shared:usage"/>
            <daml:hasValue rdf:resource="shared:production"/>
        </daml:restriction>
    </rdfs:subClassOf>
</daml:Class>
```

The definition of the concepts in our ontology enables us to compare them with concepts from other ontologies in terms of their properties and characteristics taken from the shared ontology. This comparability can be exploited information filtering in the way described in the next section.

2.4 Semantic Mapping and Filtering

Assume that we want to post a query formulated using the ontology from Fig. 2 to an information source that has been classified according to the ontology in Fig. 1. In order to answer this query, we have to resolve the heterogeneity discussed above. The use of a shared ontology in combination with a definition of the classes in both ontologies enable us to do this. As an example we take the following query (Animal $\wedge \neg$(Farm-Animal)). This query cannot be directly answered, because the term Farm-Animal is not understood, but we know what are the characteristic properties of zoo-animals and can compare them with the definitions of classes in the other ontology.

As described in the introduction, the idea of our approach is to re-write this query in such a way that it covers the same set of answers using terms from the other ontology. In general, an exact re-writing is not possible because the concepts of our ontology do not have corresponding concepts. In this case, we have to look for re-writings that approximate the query as closely as possible. Re-writings that are an upper approximation of the original query are know from the database area as *minimal subsuming mappings* [4]. While in the area of databases upper approximations are often used in combination with an additional filter that removes irrelevant results, our approach aims for correctness rather than for completeness and therefore uses a lower approximation.

The idea of the re-writing is the following. Based on the formal definitions of the classes in both ontologies, we can find those concepts in the ontology of Fig. 1 that are most closely related to a query concept. Taking a concepts from our query, we can for example decide that Domestic-Animal and Production-Animal are upper approximations for farm animals while Cow and Pig are lower approximations. Using these concepts, we can define lower boundaries for farm-animals (Cow $\vee$ Pig) and use this expression instead of the original concept still getting correct results. In our example, however, the concept occurred in a negated form. In order to return a correct result, we therefore cannot use the lower bound because not all irrelevant resources might be excluded. Based on the considerations made above we can replace the concept farm-animal within the scope of

the negation by its upper bound (Domestic-Animal ∧ Production-Animal). Using this rewriting, we get the following query that can be shown to return only correct results: (Animal ∧ ¬(Domestic-Animal ∧ Production-Animal).

The situation becomes slightly more complicated if complex expressions occur in the scope of a negation. An example is the following query: ¬(Pet ∨ Farm-Animal). In this case we first have to convert the query into negation normal form where negation only applies to atomic concepts. In negation normal form the above query will be of the form ¬Pet∧¬Farm-Animal). Using upper and lower bounds this query translates to ¬Domestic-Animal∧¬(Domestic-Animal∧ Production-animal). This query normalizes to (¬Domestic-Animal ∧ ¬Production-Animal) which in our example only includes the classes Lion and Tiger.

In the following, we show how the general idea sketched in this section can be implemented on the basis of available reasoning support for ontology languages, i.e. DAML+OIL.

3 Reasoning about Ontologies

In the overview of our approach, we argued that the use of a shared vocabulary allows us to compare concepts from different ontologies. Up to now we only gave an intuitive notion of this comparability. What we really need in order to implement the information filtering approach is the ability to automatically compare class definitions using logical reasoning. In this section, we discuss this aspect in two steps. We first review the formal semantics of DAML+OIL [14] and derive possibilities for logical reasoning we have. Afterwards we formally define the reasoning task that is implied by our approach.

3.1 Semantics of DAML+OIL

In [14] a formal semantics for DAML+OIL is described. The semantics is based on an interpretation mapping into an abstract domain. More specifically, every concept name is mapped on a set of objects, every property name is mapped on a set of pairs of objects. Individuals (in or case resources) are mapped on individual objects in the abstract domain. Formally, an interpretation is defined as follows:

Definition 1 (Interpretation) *An Interpretation consists of a pair* $(\Delta, .^{\mathcal{E}})$ *where* Δ *is a (possibly infinite) set and* $.^{\mathcal{E}}$ *is a mapping such that:*

- $x^{\mathcal{E}} \in \Delta$ *for all individual names* x.
- $C^{\mathcal{E}} \subseteq \Delta$ *for all concept names* C
- $R^{\mathcal{E}} \subseteq \Delta \times \Delta$ *for all role names* R

We call $.^{\mathcal{E}}$ *the extension of a concept, a role, or an individual, respectively.*

This notion of an interpretation is a very general one and does not restrict the set of objects in the extension of a concept. This is done by the use of operators

for defining classes. In our example, we used the `subClassOf` and the `hasValue` operator for restricting the set of objects that are members of the class zoo animals. These kinds of operators restrict the possible extensions of a concept. Figure 4 summarizes the specific interpretations of a part of the operators of DAML+OIL.

Concept forming operator	**Extension** $.^{\mathcal{E}}$
`daml:intersectionOf`	$C_1^{\mathcal{E}} \cap \cdots \cap C_n^{\mathcal{E}}$
`daml:unionOf`	$C_1^{\mathcal{E}} \cup \cdots \cup C_n^{\mathcal{E}}$
`daml:complementOf`	$\Delta - C^{\mathcal{E}}$
`daml:oneOf`	$\{x_1, \cdots, x_n\} \subset \Delta$
`daml:toClass`	$\{y \in \Delta \vert (y,x) \in P^{\mathcal{E}} \implies x \in C^{\mathcal{E}}\}$
`daml:hasClass`	$\{y \in \Delta \vert \exists x((y,x) \in P^{\mathcal{E}}) \wedge x \in C^{\mathcal{E}}\}$
`daml:hasValue`	$\{y \in \Delta \vert (y,x) \in P^{\mathcal{E}}\}$
`daml:minCardinalityQ`	$\{y \in \Delta \vert \ \vert \{x \vert (y,x) \in P^{\mathcal{E}} \wedge x \in C^{\mathcal{E}}\}\vert \leq n\}$
`daml:maxCardinalityQ`	$\{y \in \Delta \vert \ \vert \{x \vert (y,x) \in P^{\mathcal{E}} \wedge x \in C^{\mathcal{E}}\}\vert \geq n\}$
`daml:cardinalityQ`	$\{y \in \Delta \vert \ \vert \{x \vert (y,x) \in P^{\mathcal{E}} \wedge x \in C^{\mathcal{E}}\}\vert = n\}$

Fig. 4. Terminological Operators of DAML+OIL

Using this definition, we can restrict the set of all *Farm-animal* to members of the set: $\{x|x \in animal^{\mathcal{E}}\} \cap \{x|(x, domestic^{\mathcal{E}}) \in origin^{\mathcal{E}}\} \cap \{x|(x, production^{\mathcal{E}}) \in usage^{\mathcal{E}}\}$. These kinds of restriction are the basis for deciding whether a class definition is equivalent, more specialized or more general than another. Formally, we can decide whether one of the following relations between two expressions hold:

subsumption: $C_1 \sqsubseteq C_2 \iff C_1^{\mathcal{E}} \subseteq C_2^{\mathcal{E}}$
membership: $x : C \iff x^{\mathcal{E}} \in C^{\mathcal{E}}$

In order to implement information filtering, we need subsumption in order to determine the upper and lower boundaries of a concept. Membership is used in order to retrieve relevant resources that match a query.

3.2 The Reasoning Task

In order to get a clearer notion of the problem to be solved, we give an abstract definition of an information source in terms of a set of information items that are classified according to a local ontology.

Definition 2 (Information Source) *An Information source is a tuple* $\langle O, I, M\rangle$ *where* $O = \langle S, C, d\rangle$ *is a local ontology with shared ontology* S*, a set of class names* C *and a mapping* d *that assigns a DAML+OIL class definition over terms from* S *to every class name from* C*,* I *is a set of information items and* $M : I \times C$ *is a membership relation relates information items to classes of the source ontology.*

Building on this abstract view on an information source, we can define the problem of integrating the classifications employed in two different systems. Roughly speaking the task is to extend the membership relation M_1 of an information source IS_1 by an additional relation M' that relates the information items of a second information source IS_2 according to the source ontology of IS_1.

Definition 3 (Integration Problem) *Let* $IS_1 = \langle\langle S, C_1, d_1\rangle, I_1, M_1\rangle$ *and* $IS_2 = \langle\langle S, C_2, d_2\rangle, I_2, M_2\rangle$ *be information sources, then a bilateral integration problem is equivalent to finding a membership relation* $M : I_1 \cup I_2 \times C_1$ *such that for all* $x \in I_2 \cup I_2$ *and* $c_i \in C_1$*:*

$$(x, c_i) \in M \text{ iff } x^{\mathcal{E}} \in d_1(c_i)^{\mathcal{E}}$$

In order to find this new relation M we have to rely on the semantics of both information sources that is given by their source ontology. The use of the same shared ontology provides us with a unique interpretation of the primitive terms defined therein. These terms are used in the source ontologies to define different sets of more complex concepts. The use of DAML+OIL as a standard language allows use to reason across the different source ontologies.

4 Approximate Information Filtering

In an ideal case, we will have enough information about the resources in both ontologies in order to directly check the membership of a resource with respect to concepts in our own ontology. In many cases, however, we will only have the assignment of resources to classes in their own local ontology. In this case, we have to rely on an approximate comparison of concepts in both ontologies as described in the overview of the method. In this section, we discuss the use of approximations of concepts in other ontologies and the use of these approximations for query re-writing on a formal level.

4.1 Re-classification

Consider the situation where we want to classify an information item from an information source IS_2 into the local ontology of IS_1 by computing M. The only information we have about x is its classification M_2 with respect to the source ontology of IS_2. In order to make use of this information, we have to determine the relation between possible classifications of x in IS_1 and the source ontology of IS_2. In this context, we can use subsumption testing in order to determine hypotheses for M with respect to IS_2 by computing the class hierarchy for $C_1 \cup C_2$ using the definitions of individual classes.

As the classes in the hierarchy form a partial order, we will always have a set of direct super- and a set of direct subclasses of c_1. We can use these classes as upper and lower approximation for c_1 in IS_2:

Definition 4 (Upper Approximation) *Let* $IS_1 = \langle\langle S, C_1, d_1\rangle, I_1, M_1\rangle$ *and* $IS_2 = \langle\langle S, C_2, d_2\rangle, I_2, M_2\rangle$ *be information sources and* $c \in C_1$ *a class from* IS_1*, then a class* $c_{lub} \in C_2$ *is called a least upper bound of* c *in* IS_2*, if the following assertions hold:*

1. $d_1(c) \sqsubseteq d_2(c_{lub})$
2. $(\exists c' \in C_2 : d_1(c) \sqsubseteq d_2(c')) \implies (d_2(c_{lub}) \sqsubseteq d_2(c'))$

The upper approximation $lub_{IS_2}(c)$ *is the set of all least upper bounds of* c *in* IS_2*.*

Definition 5 (Lower Approximation) *Let* $IS_1 = \langle\langle S, C_1, d_1\rangle, I_1, M_1\rangle$ *and* $IS_2 = \langle\langle S, C_2, d_2\rangle, I_2, M_2\rangle$ *be information sources and* $c \in C_1$ *a class from* IS_1*, then a class* $c_{glb} \in C_2$ *is called a greatest lower bound of* c *in* IS_2*, if the following assertions hold:*

1. $d_2(c_{glb}) \sqsubseteq d_1(c)$
2. $(\exists c' \in C_2 : d_2(c') \sqsubseteq d_1(c)) \implies (d_2(c') \sqsubseteq d_2(c_{glb}))$

The lower approximation $glb_{IS_2}(c)$ *denotes the set of all greatest lower bounds of* c *in* IS_2*.*

The rational of using these approximations is that we can decide whether x is a member of the classes involved based on the relation M_2. This decision in turn provides us with an approximate result on deciding whether x is a member of c_1, based on the following observation:

- If x is member of a lower bound of c_1 then it is also in c_1
- If x is not member of all upper bounds of c_1 then it is not in c_1

In [12] Selman and Kautz propose to use this observation about upper and lower boundaries for theory approximation. We adapt the proposal for defining an approximate classifier $M' : I_2 \times C_1 \to \{0, 1, ?\}$ in the following way:

Definition 6 (Approximate Re-classification)
Let $IS_1 = \langle\langle S, C_1, d_1\rangle, I_1, M_1\rangle$ *and* $IS_2 = \langle\langle S, C_2, d_2\rangle, I_2, M_2\rangle$ *be information sources and* $x \in I_2$ *then for every* $c_1 \in C_1$ *we define* M' *such that:*

- $M'(x, c_1) = 1$ *if* $x : \left(\bigvee_{c \in glb_{IS_2}(c_1)} d_2(c) \right)$
- $M'(x, c_1) = 0$ *if* $x : \neg \left(\bigwedge_{c \in lub_{IS_2}(c_1)} d_2(c) \right)$
- $M'(x, c_1) = ?$*, otherwise*

Where the semantics of disjuction and conjunction is defined in the obvious way using set union and intersection .

Based on the observation about the upper and lower bounds, we can make the following assertion about the correctness of the proposed approximate classification:

Proposition 1 (Correctness of the Approximation) *The approximation from definition 6 is correct in the sense that:*

1. *If* $M'(x, c_1) = 1$ *then* $x^{\mathcal{E}} \in d_1(c_1)^{\mathcal{E}}$
2. *If* $M'(x, c_1) = 0$ *then* $x^{\mathcal{E}} \notin d_1(c_1)^{\mathcal{E}}$

Using the definition of upper and lower bounds the correctness of the classification can be proven in a straightforward way:

Proof. (1) If the classification returns $M'(x, c_1) = 1$ then $x : (\bigvee_{c \in glb_{IS_2}(c_1)} d_2(c))$. Using definition 5 we get that for all c we have $d_2(c) \sqsubseteq d_1(c_1)$ and therefore also $(\bigvee_{c \in glb_{IS_2}(c_1)} d_2(c)) \sqsubseteq d_1(c_1)$ (by set theory). Using the definition of subsumption we can conclude that $x^{\mathcal{E}} \in d_1(c_1)^{\mathcal{E}}$.

(2) Using definition 4 we deduce that for all c we have $d_1(c_1) \sqsubseteq d_2(c)$ and therefore $d_1(c_1) \sqsubseteq \bigwedge_{c \in lub_{IS_2}(c_1)} d_2(c)$. This means that $x^{\mathcal{E}} \in d_1(c_1)^{\mathcal{E}}$ only if $x^{\mathcal{E}} \in (\bigwedge_{c \in lub_{IS_2}(c_1)} d_2(c))^{\mathcal{E}}$. However if the classification returns $M'(x, c_1) = 0$ then $x : \neg(\bigwedge_{c \in lub_{IS_2}(c_1)} d_2(c))$ which is equivalent to $x^{\mathcal{E}} \notin (\bigwedge_{c \in lub_{IS_2}(c_1)} d_2(c))^{\mathcal{E}}$. Therefore we also have $x^{\mathcal{E}} \notin d_1(c_1)^{\mathcal{E}}$.

This results provides us with the possibility to include many of the information items from remote systems into an information source in such a way, that we get a semantic description of the item we can use for information management. Another interesting application of this approach, namely information filtering is described in the next section.

4.2 Query Re-writing

The considerations from last section provide a formal basis for query re-writing. Having proven the correctness of the approximation we can use them to re-write queries by replacing concept names by their approximation. This re-writing is part the overall filtering process that consists of the following steps:

1. **Normalization:** the original query is transformed into negation normal form.
2. **Re-writing:** the concept names in the query are replaced by their approximations in the remote source
3. **Classification:** the re-written query is classified into the ontology of the remote source and instances of subsumed concepts are returned as result.

A query is said to be in negation normal form if negations only apply to single concept names and not to compound expressions. We can only re-write queries in normal form, because we have to distinguish between non-negated concept names that are replace by their lower approximation and negated ones replaced by the upper approximation. Once we have transformed the query into negation normal form, we can re-write it in the described way. Formally this re-writing can be defined as follows:

Definition 7 (Query Re-writing) *The rewriting of a query Q over concepts c_i from an information source IS_1 to a query Q' over concepts from another information source IS_2 is defined as as follows:*

- *replace every non negated concept name c by:* $\bigwedge_{c' \in lub_{IS_2}(c)} c'$
- *replace every negated concept name c by:* $\bigvee_{c' \in glb_{IS_2}(c)} c'$

The rewriting and execution of a query can easily be implemented using the Description Logic System RACER [8]. We can compute the re-writing using Algorithm 1. The input for the algorithm is the query to be re-written, the class names in C_2 and a DAML+OIL knowledge base including the definitions of the concepts in C_1 and C_2 as well as the shared ontology.

Algorithm 1. Rewrite-Query

Require: A Boolean query in negation normal Form: Q
Require: A list of class names: N
Require: A terminological knowledge base T
  racer.in-tbox(T)
  for all t is an atomic term in Q **do**
    if t is negated **then**
      $B[t]$:= racer.directSupers(t)
      $B'[t] := B[t] \cap N$
      $Q(t) := (c_1 \wedge \cdots \wedge c_n)$ for $c_i \in B'[t]$
    else
      $B[t]$:= racer.directSubs(t)
      $B'[t] := B[t] \cap N$
      $Q(t) := (c_1 \vee \cdots \vee c_n)$ for $c_i \in B'[t]$
    end if
    Q' := **proc** Replace t in Q' by $Q(t)$
  end for
  return Q'

As the re-writing builds upon the approximations discussed in the last section we can guarantee that the result of the query is correct. Moreover, we can use

subsumption reasoning in order to determine this result. To be more specifically, a resource x is indeed a member of the query concept if membership can be proved for the re-written query.

Proposition 2 (Correctness of re-writing) *The notion of query re-writing defined above is correct in the sense that:*

$$x : Q' \implies x^{\mathcal{E}} \in Q^{\mathcal{E}}$$

Proof. From proposition 1 we get that $x : (\bigwedge_{c' \in lub_{IS_2}(c)} c')^{\mathcal{E}}$ implies $x^{\mathcal{E}} \in c^{\mathcal{E}}$ and that $x : \neg(\bigvee_{c' \in glb_{IS_2}(c)} c')^{\mathcal{E}}$ implies $x^{\mathcal{E}} \notin c^{\mathcal{E}}$. This establishes the correctness of re-writing for atomic queries, i.e. non negated and negated concept names. Assuming a queries in negation normal form, it remains to be shown that the correctness is preserved for conjunctions and disjunctions of negated and non-negated concept names. This can easily be shown by induction over the definition of Boolean query expressions.

This correctness result enables us to implement the information filtering approach using available subsumption reasoner in combination with a query-processor that re-writes queries and poses the resulting query to these reasoners. A simplified version of this approach has already been implemented in an information brokering system developed at the University of Bremen.

5 Related Work

The idea of rewriting queries based on the special capabilities of a remote system is first reported in [10]. It has been applied in the area of information retrieval [5] to translate full-text queries and in database systems for translating SQL queries [4]. The use of terminological reasoning for database schema integration is discussed in [3]. The approach has also been adapted for the problem of integration ontologies [2]. Both approaches are based on the mapping rather than a translation approach.

6 Discussion

In this paper we discussed the topic of using semantic web technology, i.e. ontologies for information filtering. We argued that the use of concept expressions instead of keywords will improve precision and recall and is therefore a promising approach. In the remainder of the paper we discussed problems that arise from the need to use different possibly heterogeneous ontologies for this purpose and argued that existing approaches to ontology alignment, i.e. merging and mapping have drawbacks in terms of flexibility and integration effort. As the main contribution of the paper, we proposed an approach to ontology alignment for information filtering that is based on query re-writing in order to adapt Boolean

queries to the vocabulary of an other ontology that is motivated by research in the area of information systems and databases.

Our approach was based on two assumptions: (1) the existence of a shared ontology in terms of a RDF scheme model and (2) the ability to perform subsumption and membership checking in the ontology language used to encode local ontologies of different information sources. We argued that the formal semantics of DAML+OIL provides the latter and can be used to implement information filtering making use of existing logical reasoners. We showed that queries can be re-written in such a way that we can proof correctness of the results. A questions that are left open is how we can provide the ontological infrastructure needed to apply the method. In previous work, we described how parts of such an infrastructure can be build [13], however the design of suitable shared vocabularies is still an open problem.

Considering the task of information filtering we might also take into consideration re-writings that are complete but not correct in order to avoid that we loose important information. In this case, however, we need to make sure that the number of irrelevant documents does not become too large, because this would mean loosing the advantage of concept based filtering.

References

1. N.J. Belkin and B.W. Croft. Information filtering and information retrieval: Two sides of the same coin? *Communications of the ACM*, 35(12):29–38, December 1992.
2. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. A framework for ontology integration. In *Proceedings of the international semantic web working symposium*, Stanford, USA, 2001.
3. Diego Calvanesea, Giuseppe De Giacomo, and Maurizio Lenzerini. Description logics for information integration. In *Computational Logic: From Logic Programming into the Future*, Lecture Notes in Computer Science. Springer Verlag, 2001.
4. K.C.-C. Chang and H. Garcia-Molina. Approximate query mapping: Accounting for translation closeness. *The VLDB Journal*, 10:155–181, 2001.
5. K.C.-C. Chang, H. Garcia-Molina, and A. Paepcke. Boolean query mapping across heterogeneous information sources. *IEEE Transaction on Knowledge and Data Engineering*, 8(4), 1996.
6. William B. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms*. Prentice-HALL, North Virginia, 1992.
7. R. Gaizauskas and K. Humphreys. Using a semantic network for information extraction. *Journal of Natural Language Engineering*, 1997.
8. Volker Haarslev and Ralf Moller. Description of the RACER system and its applications. In *Proceedings of the Description Logics Worlshop DL-2001*, Stanford, CA, 2001.
9. Michel Klein. Combining and relating ontologies: an analysis of problems and solutions. In *Ontologies and information sharing*, number 47, Seattle, USA, August 2001.
10. Yannis Papakonstantinou, Ashish Gupta, and Laura Haas. Capabilities-based query rewriting in mediator systems. In *Proceedings of 4th International Conference on Parallel and Distributed Information Systems*, Miami Beach, Flor., 1996.

11. G. Salton and M.J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, 1983.
12. B. Selman and H. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, March 1996.
13. H. Stuckenschmidt and F. van Harmelen. Ontology-based metadata generation from semi-structured information. In *Proceedings of the first intenational conference on knowledge capture (K-CAP'01)*. Sheridan Printing, 2001.
14. Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. A model-theoretic semantics for daml+oil (march 2001). `http://www.daml.org/2001/03/model-theoretic-semantics.html`, march 2001.
15. Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. Reference description of the daml+oil (march 2001) ontology markup language. `http://www.daml.org/2001/03/reference.html`, march 2001.
16. Pepjijn R. S. Visser, Dean M. Jones, T. J. M. Bench-Capon, and M. J. R. Shave. An analysis of ontological mismatches: Heterogeneity versus interoperability. In *AAAI 1997 Spring Symposium on Ontological Engineering*, Stanford, USA, 1997.
17. David Yarowsky. Word-sense disambiguation using statistical models of Roget's categories trained on large corpora. In *Proceedings of COLING-92*, pages 454–460, Nantes, France, 1992.

ParleE: An Adaptive Plan Based Event Appraisal Model of Emotions

The Duy Bui, Dirk Heylen, Mannes Poel, and Anton Nijholt

University of Twente
Department of Computer Science
The Netherlands
{theduy,heylen,mpoel,anijholt}@cs.utwente.nl

Abstract. We propose ParleE, a quantitative, flexible and adaptive model of emotions for a conversational agent in a multi-agent environment capable of multi-modal communication. ParleE appraises events based on learning and a probabilistic planning algorithm. ParleE also models personality and motivational states and their role in determining the way the agent experiences emotion.

1 Introduction

In this paper, we describe ParleE, a quantitative, flexible and adaptive model of emotions for an embodied conversational agent situated in a multi-agent environment that can engage in multi-modal communication. ParleE is developed in order to enable the conversational agent to respond to events with the appropriate expressions of emotions with different intensities.

Research in our Parlevink group at the University of Twente has focused on human-computer interaction in virtual environments ([7], [8]). Our aim is to build believable agents for several application areas: information, transaction, education, tutoring and e-commerce. For embodied agents to be believable, the minds of agents should not be restricted to model reasoning, intelligence and knowledge but also emotions and personality. Furthermore, it is necessary to pay attention not only to the agent's capacities for natural language interaction but also to its non-verbal aspects of expression. We propose ParleE for this purpose.

ParleE is built upon existing computational models of emotions: Elliott's Affective Reasoner [3], Velásquez's Cathexis [11], El-Nasr et al's FLAME [4] and Gratch's Émile [6]. Like many of these and other models, ParleE generates emotions based on Ortony et al.'s theory of appraisal [9]. ParleE appraises events based on learning and a probabilistic planning algorithm. We also pay attention to the integration of personality and motivational states into ParleE.

Section 2 will describe the requirements for ParleE and compare how existing models suit these requirements. Section 3 will discuss the ParleE model. Section 4 will show how to calculate the intensity of emotions. The model of personality will be discussed in Sect. 5. Finally, the learning components and an illustration of the model are presented in Sects. 6 and 7 respectively.

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 129–143, 2002.

2 From Requirements, Existing Models to ParleE

To be consistent with the aim of building embodied agents with different personalities for various application areas, we introduce a set of requirements for our model. We then give a brief review of existing models to see how they can fit in our requirements.

2.1 Requirements

The requirements for the proposed system are:

Quantitative Model: The model should produce emotions with different intensities in an agent. The emotion intensities should also decay over time.

Flexible Appraisal Model: The model should be an event appraisal model. Moreover, the model should have a generic, domain-independent way of appraising events to be useful in a variety of applications.

Incorporation of Personality: Personality plays an important role in determining the intensity of emotions and should therefore be incorporated in a model of emotions.

Incorporation of Motivational States: Motivational states such as pain and hunger should be included because they also influence the emotion system.

Adaptability: The model should be able to learn to reflect the dynamic status of the environment.

2.2 Existing Computational Models of Emotions

Many models of emotions have been proposed for various purposes with differences in focus. For a survey, see [4]. We will now study how models, that inspire ParleE, suit our requirements.

2.3 Cathexis Model

Cathexis is proposed by Velásquez [11]. By considering both cognitive and non-cognitive elicitors of emotions such as sensorimotor and motivational states, Cathexis models six basic emotions: anger, fear, distress/sadness, enjoyment/happiness, disgust and surprise. Cathexis also differentiates emotions from other kinds of affective phenomena, such as mood. The intensity of emotion e at time t is calculated based on the intensity of emotion e at time $t-1$ and the values of emotion e's elicitors. The value of an elicitor of emotion e is derived from the value and weight of conditions that contribute to the activation of the elicitor. This model is limited because these values and weights have to be predefined. Therefore, it is not a flexible model. Moreover, the system is not adaptive and there is no integration of personality.

2.4 Elliott's Affective Reasoner

Elliott's Affective Reasoner [3] is a computational adaptation of the OCC model [9]. This model assesses the relationship between events and an agent's disposition (described by its goals, social standards, and preferences). The relationship is characterized in terms of a set of features called *emotion-eliciting conditions*. Elliott's Affective Reasoner is not a quantitative one as it does not consider the intensities of emotions. Besides, there is no adaptation, integration of personality and motivational states in this model. This model is also limited by the use of domain-specific rules to appraise events.

2.5 FLAME

El-Nasr et al's FLAME model [4] is a computational model of emotions based on event appraisal. It incorporates some learning components to increase the adaptation in modelling emotions. It also uses an emotion filtering component, which takes into account motivational states, to resolve conflicting emotions. FLAME uses fuzzy logic rules to map assessments of the impact of events on goals into emotional intensities. However, the agent in FLAME does not have clearly defined goals. Moreover, the model does not provide a way of calculating the impact of an event on an agent's goal. Instead, it uses a predefined reward value for the user's action's impact on an agent's goal. Hence, this is an inflexible model. Personality is also not mentioned in this model.

2.6 Gratch's Émile

Gratch's Émile [6] uses classical planning methods (detecting and resolving threats) to appraise the emotional significance of events as they relate to plans and goals, to model and predict the emotional state of other agents, and to alter behavior accordingly. In this model, Gratch has proposed a simple way of calculating the probability of a goal to be achieved, the probability of threats to a goal and its importance. Standards are defined as domain-specific constraints of behavior such as "thou shalt not kill". The limitation of this model is that it leaves out the value of an event's unexpectedness when calculating the intensity of event-based emotions like joy and distress. Moreover, Émile's threats detection approach would mistreat the event that is both establisher and threat to the agent's goal. For example, for an agent having a goal of watching TV, the preconditions (or subgoals in Émile) are the TV is in the living room and the TV is not broken. However, the TV currently is broken although it is in the living room. An available plan to achieve the top-level goal are "bring the TV to the shop to have it fixed", and then "bring the TV back". In Émile, the action of "bring the TV to the shop" is to satisfy the second condition/subgoal that the "the TV is not broken", but is considered as a threat to the first condition/subgoal that "the TV is in the living room". Émile would generate sadness for the event "the TV is in the shop", which is not logically sensible. Our approach, which uses a probabilistic planning algorithm with heuristic searching, does not encounter this problem. We do not appraise an event by considering it as an establisher or a threat to the agent's goal but by assessing how that event changes the probability of achieving the agent's goal.

Émile also does not pay attention to the way motivational states and personality influence emotion.

3 The ParleE Model

In this section, we will give an overview of ParleE. Its components will then be discussed in more detail.

3.1 Overview

When an event happens, an Emotion Impulse Vector (EIV) is generated by appraising the event using the rules proposed by the OCC appraisal theory [9] based on the agent's goals, plans and standards. An EIV contains the values of the event's impact on emotions. The EIV is then used to update the Emotion State Vector (ESV), that contains values representing intensities of emotion. This will be discussed in Sect. 4.4. An overview of the system can be seen in Fig. 1.

The Emotion Appraisal Component takes the event, personality, plan, and models of other agents as inputs to produce the EIV as output. This component will be explained in Sect. 3.2.

The Planner produces a plan to achieve the agent's goal. It also calculates the probability of achieving the goal. This probability is then used by the Emotion Appraisal Component to calculate the EIV. The planner is discussed in Sect. 3.3.

The Emotion Component takes the EIV and motivational states as inputs and produces the updated emotion vector as the output. This component also cooperates with the Emotion Decay component to produce decayed emotions. This component is described in Sect. 3.4.

The Emotion Decay Component calculates how emotions decay taking into account the personality. The decay function is discussed in Sect. 4.3.

Models of other agents are used to generate **Desire-other** emotions (emotions about the fortunes of others) and to predict other agents' behavior. They will be explained in Sect. 3.6.

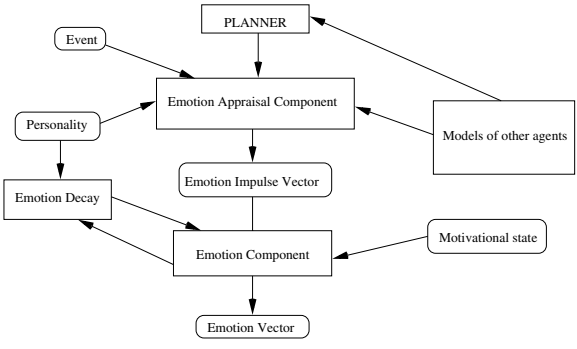


Fig. 1. Overview of the system

3.2 Emotion Appraisal Component

The Emotion Appraisal Component is in charge of generating emotions when an event occurs. This is done based on Ortony et al.'s appraisal theory [9]. Emotions are generated by assessing events relating to goals, expectations and standards.

ParleE appraises events based on learning and a probabilistic planning algorithm. For relating the events to standards, we also use a learning component. For assessing the significance of an event to the agent's goal, we make use of a probabilistic planning algorithm. In a probabilistic planning algorithm, the aim is to find a plan with the optimal probability of achieving the goal. We take the difference between the probability of achieving the agent's goal before and after the event happens as the effect of the event on the goal.

3.3 The Planner

We extend Blum and Langford's Probabilistic GraphPlan algorithm [1] to support planning in a multi-agent environment. Probabilistic GraphPlan uses graph structure to solve STRIPS-style planning problems. It allows the calculation of a plan's probability of success from a specific state of the world. We then take the change in a goal's probability of success when an event happens as the impact of the event on the goal.

In STRIPS domains [5], the state of the world is a set of true facts. A *goal* is a set of facts that we want to be true. An *operator* represents a legal action that may be performed. An operator has conjunctive preconditions, a list of facts to be added and a list of facts to be deleted. Blum and Langford extend STRIPS to have probabilistic actions. Each operator then has conjunctive preconditions and a set of possible outcomes. Each outcome is defined as a set of add and delete effects, each set having an associated probability. For example, a cook action might require that there is food, and with 80% probability of having a good meal, and 20% probability of having a bad meal.

To support planning in a multi-agent environment, we incorporate models of other agents (see Sect. 3.6) to predict other agents' behavior. We denote the probability for an agent a to achieve his/her goal at time t and the state of the world s as $P_{goal}(a, t|s)$. Provided it is agent b's turn to act, $P_{goal}(a, t|s)$ is calculated as:

$$P_{goal}(a, t|s) = \sum_{i=1,n} P(b, action_i, t|s) \times P_{goal}(a, t|s, action_i)$$

where $P(b, action_i, t|s)$ is the probability of an agent b to perform an action i at time t and the state of the world s, $P_{goal}(a, t|s, action_i)$ is the probability for an agent a to achieve his/her goal at time t and the state of the world s provided action i is performed. $P_{goal}(a, t|s, action_i)$ is calculated as:

$$P_{goal}(a, t|s, action_i) = \sum_{s' \in succ(s)} P(s'|action_i, s) \times P_{goal}(a, t + 1|s')$$

where s' is a possible state of the world with probability of $P(s'|action_i, s)$ if action i is performed at the state of the world s.

We use probabilistic approach to select an agent's action to avoid local minima. For an agent a, an action i is selected with probability:

$$P(a, action_i, t|s) = \begin{cases} \frac{P_{goal}(a,t|s,action_i)}{\wp} & \text{if } \wp > 0 \\ \frac{1}{n} & \text{if } \wp = 0 \end{cases}$$
$$\wp = \sum_{j=1,n} P_{goal}(a, t|s, action_j)$$

We also extend this planning algorithm to support multiple goals. Each goal is then associated with a priority value (from 0.0 to 1.0). The goal with the highest priority will be planned first.

3.4 The Emotion Component

The Emotion Component contains a representation of emotions and is responsible for updating the emotional state. Emotions are represented as a vector of intensities for every emotion type. Each emotion is associated with two thresholds, which will be described below.

As in Cathexis [11], we distinguish between moods and emotions by the level of arousal. We adopt Cathexis's concept of two thresholds associated with each emotion type. The first threshold, denoted as α, controls the activation of the emotion type and differentiates between mood and emotion. Emotion types with intensity level lower than their α are considered as moods, and do not have as much influence as emotions on an agent's behavior. The second threshold, denoted as ω, controls the saturation of the emotion type (the upper limit for intensity of the emotion type).

3.5 Motivational State

As in [4], we consider hunger, fatigue, thirst, and pain as motivational states. In ParleE, motivational states influence emotions by changing the α threshold for each emotion. When the level of *fatigue* gets higher, the α thresholds for negative emotions decrease (when the agent is tired, his/her negative emotions seem to be easily activated) and the α thresholds for positive emotions increase. High levels of pain tend to decrease the α threshold for emotion fear, while high levels of hunger and thirst tend to increase the α thresholds for all emotions.

3.6 Models of Other Agents

In our system, the agent has also models of other agents' goals and plans (we consider the user as an agent as well) in order to predict other agents' behavior and in order to appraise **Desire-other** emotions (emotions about the fortunes of others). The goals of other agents can be known through communication. Based on that information, the agent can predict other agents' plans. The probability that another agent performs an action is derived from the predicted plan. To appraise emotions about another agent's fortunes, the desirability of an event to another agent's goals is derived from the predicted plan.

4 Intensity of Emotions

To compute the intensities of emotions, Ortony et al. [9] have proposed several intensity variables including *desirability*, *praiseworthiness*, *appealingness*, and *unexpectedness*. Based on that, we propose a quantitative model that uses six variables: *impact of an event on goal* and *goal importance* (to compute the *desirability*), *probability of achieving a goal* and *probability of an event occurring* (these two probabilities are equivalent to the *unexpectedness*), *praiseworthiness* (value of an action), and *liking* (between agents). These variables are used to calculate the EIV rather than the emotion vector. For each emotion type, we first calculate the value of the impulse with regard to each goal. The sum of all impulses for all goals is taken as the final impulse for that emotion type. The impulse vector is then used to update the emotion vector.

We will now discuss these emotional variables, followed by the EIV, the decay function and the integration of the EIV into the emotion vector.

4.1 Emotion Variables

In this section, we describe how to calculate the value of emotion variables. These variables are used by the Emotion Appraisal Component to calculate the EIV, which will be combined with the current emotion vector to generate a new emotion vector.

Goal importance is an important factor for determining the intensity of goal-related emotions. It is denoted as $Import(goal)$. In ParleE, we only consider top-level goals rather than any subgoals that arise in the plans developed to achieve top-level goals. Thus, if an event happens that affects the goal but does not make the goal succeed or fail, we consider that it partially affects the goal. Because of this approach, we are only concerned with the intrinsic importance of the goal (as defined in [6] as "the reward (utility) an agent receives from achieving the goal"). We do not consider a goal's extrinsic worth (how a goal promotes other goals) as we do not have subgoals in ParleE. For the intrinsic importance of the goal, we assign predefined values as in [6].

The probability of achieving a goal, denoted as $P(goal)$, is adopted from Blum and Langford's Probabilistic GraphPlan algorithm. This algorithm computes the optimal probability of a goal success from a state of the world. We take that value as the probability of achieving the goal. The advantage of a plan based emotion model over other approaches is that the planning algorithm allows a generic (domain-independent) way of calculating this probability.

The probability of an event occurring, which is denoted as $P(event)$, is defined in ParleE as follows:

- If the agent is waiting for the result of his/her own action then $P(event)$ is the probability of an outcome of an action (c.f. Sect. 3.3). This probability is first assigned some predefined value. It is then updated by learning from what actually happens, which will be discussed in Sect. 6.

- If the agent is waiting for another agent's action then $P(event)$ is the probability of another agent's action at the current world state s times the probability of an outcome of that action. The probability of another agent's action is calculated from this agent's model of another agent (c.f. Sect. 3.3 and 3.6).

The impact of an event on a goal in this model is the difference between the probability of achieving the goal before and after the event happens (denoted as *Impact(event,goal)*). Thus, if the value of impact is negative, that means the probability decreases after the event happens, the event is undesirable for this goal; if the value of impact is positive, that means the probability decrease after the event happens, the event is desirable for this goal.

$$\textit{Impact(event,goal)} = P_{\textit{after}}(\textit{goal}) - P_{\textit{before}}(\textit{goal})$$

Praiseworthiness is used to evaluate how the agent's action meets standards of behaviors. We use the idea of FLAME [4] about learning values of actions to form the standards in appraising emotions. The variable is denoted as *Praiseworthiness(action)*. Another agent can provide feedback on the agent's action. In ParleE, the feedback is a point which ranges from -1.0 to 1.0. The average feedback point for an action over the time is taken as the value of *Praiseworthiness(action)*.

Liking, denoted as $\textit{LikingLevel}(\textit{another agent})$, is a variable that contributes to the intensity of **Desire-other** emotions. In ParleE, the agent maintains a dynamic liking level towards each another agent. Each liking level ranges from -1.0 to 1.0. Starting with a neutral attitude toward another agent (the value of liking level is 0.0), the agent's liking level changes when another agent has performed an action that affected the agent's goal. The agent's liking level increases if it is a desirable event and decreases if it is undesirable. The agent's liking level towards another agent is updated as follow if the preceding event is caused by another agent:

$$\textit{Desirability(event)} = \sum_{\textit{all goal}} \textit{Import(goal)} \times \textit{Impact(event,goal)}$$

$$\textit{LikingLevel}_{t+1} = max(-1.0, min(1.0, \textit{LikingLevel}_t + 0.1 * \textit{Desirability(event)}))$$

4.2 The Emotion Impulse Vector

We now present formulas to calculate the intensity of some emotion impulses. For other emotion impulses, it works the same way based on the value of all variables described above.

Hope arises from the belief that a goal is going to succeed and it is important that the goal does not fail. So **Hope** arises only when the probability of the goal to succeed is higher than 0.5. On the contrary, **Fear** arises from the belief that a goal is going to fail (probability of achieving the goal is lower than 0.5). Therefore, the probabilities of the goals to succeed directly relate to the intensity of **Hope** and **Fear**. In addition, the probability of achieving a more important goal seems to have more influence on the intensity of **Hope** and **Fear**. To capture those characteristics of **Hope** and **Fear**, we propose the following formulas:

$$\textit{Hope} = \sum_{\textit{all goal}} \textit{Import(goal)} \times (P(\textit{goal}) - 0.5)$$

$Fear = \sum_{all\ goal} Import(goal) \times (0.5 - P(goal))$

Happiness arises when a goal succeeds or becomes more likely to succeed (the value of the impact from the event on the goal is positive). **Sadness** arises when the goal is more likely to fail (the value of the impact from the event on the goal is negative). The value of the impact of an event and the importance of the goal are the two factors contributing to the intensity of **Happiness** and **Sadness**. Moreover, the unexpectedness of the event also plays a role in determining the intensity of the two emotions, and it is likely to have more effect on **Happiness** than **Sadness**. The intensities of **Happiness** and **Sadness** impulses are modelled as follows:

$Happiness = \sum_{all\ goal} Impact(event,goal) \times Import(goal) \times \sqrt{1 - P(event)}$

$Sadness = \sum_{all\ goal} Impact(event,goal) \times Import(goal) * (1 - P(event))^2$

Anger arises when another agent performs some action that is undesirable for the agent. The intensity of **Anger** is also calculated using the value of the impact of the event , the importance of the goal and the probability of the event to happen:

$Anger = \sum_{all\ goal} Import(goal) \times \sqrt{1 - P(event)} \times Impact(event,goal)$

Surprise arises when some unexpected event happens. In ParleE, **Surprise** arises only when $P(event) < 0.5$. The intensity of the **Surprise** impulse is considered as the unexpectedness of the event:

$Surprise = 0.5 - P(event)$

The **Happy_for** emotion arises when an agent that is liked is happy (when the value of liking level is positive). We consider this emotion as an example of **Desire-other** emotions. The intensity of **Happy_for** is determined by the agent's liking level towards another agent and the intensity of another agent's **Happiness** (derived from the models of other agents):

$Happy\_for = LikingLevel(another\ agent) \times (another\ agent's\ Happiness)$

Pride and **Shame** are two emotions related to standard of behaviors. The value of these depends on the praiseworthiness of the performed action (if the praiseworthiness is positive then **Pride** arises, if it is negative then **Shame** arises):

$Pride = Praiseworthiness(action)$

$Shame = -Praiseworthiness(action)$

4.3 Decay Function

It is important to derive a reasonable decay function of emotions over time. It seems that emotions decay slower when their intensity levels are lower. With the definition of mood above, it is reasonable that moods persist much longer than high intensity emotions. Moreover, negative emotions tend to decay slower than positive emotions. Personality also influences how emotions decay. Taking into account these considerations, we propose a decay function as follows:

$\Psi(E_{i,t+1}) = \Phi(E_i, personality) \times E_i(t)$

where $E_i(t)$ is the intensity of an emotion E at time t, $\Phi(E, personality)$ is a function of emotions that generates different decay rates for each emotion with regard to the agent's personality. In ParleE, the *Inclination Level* of the agent's *feeling emotions* (see table 1 in Sect. 5) will affect the function $\Phi(E, personality)$. The higher the *Inclination Level* of the agent's *feeling emotions* is, the slower the emotions decay. The decay rate for an emotion E_i is modified as follows:

$\Phi(E_i, personality) = decayFactor(E_i) * feelingLevel$

4.4 Updating Emotions

To update emotions over time, one has to consider the previous emotion states, the decay function, and the values of emotion impulses.

The intensity of emotion i is the decayed value of the intensity at previous time plus the reduced value of emotion impulse i. As the value of emotion impulse i is affected by the intensities of emotions at previous time, it is proportionally reduced by the sum of effect values from all emotions to emotion i. Finally, the intensity of emotion i is limited to the upper threshold ω_i by the function min:

$$E_{i,t+1} = min(\omega_i, \Psi(E_{i,t}) + EI_i \times (1 - \sum_j \frac{S_{j,t}}{\omega_j} \times M_{j,i}))$$

$$S_{j,t} = \begin{cases} 0 & \text{if } E_{j,t} < \alpha_j \\ E_{j,t} & \text{if } E_{j,t} \geq \alpha_j \end{cases}$$

where $E_{i,t}$ is the intensity of emotion i at time t, ω_i is the upper threshold for the intensity of emotion i , $\Psi(E_{i,t})$ is the decay function as described above, EI_i is the intensity of emotion impulse for emotion i, $M_{j,i}$ is the effect factor from emotion j to emotion i. To assure that the effect of an impulse on an emotions has as minimum 0 and as maximum the intensity of the impulse itself, we introduce the following constraint on $M_{j,i}$:

$0 \leq \sum_j M_{j,i} \leq 1 \quad \forall i$

Since $\quad 0 \leq E_{j,t} \leq \omega_j \quad \forall j$

$\implies \quad 0 \leq S_{j,t} \leq \omega_j \quad \forall j$

$\implies \quad 0 \leq \frac{S_{j,t}}{\omega_j} \leq 1 \quad \forall j$

$\implies \quad 0 \leq \frac{S_{j,t}}{\omega_j} \times M_{j,i} \leq M_{j,i} \quad \forall j$

$\implies \quad 0 \leq \sum_j \frac{S_{j,t}}{\omega_j} \times M_{j,i} \leq 1$

5 Model of Personality

5.1 Rousseau's Model

We use Rousseau's model of personality [10] as it presents a clear view on the relevant dimensions of a personality. It also provides "a sufficiently rich structure based on convention architecture of an intelligent agent" [10]. Personalities are classified according to different processes that an agent can perform: perceiving, reasoning, learning, deciding, acting, interacting, revealing, and feeling emotions. Each process is considered at two levels: the natural inclination that the agent has to perform the process, and the main aspect that the agent focuses on while performing the process. A summary of the

dimensions of a personality can be seen in table 1. We found this model convenient and easy to implement and to assess the influence of personality on other processes (eg. emotion). We also extended this qualitative model to a quantitative model. A personality is now represented as a vector in 16-dimensional space:

Personality $= (p_1, p_2, ...p_{16})$ where $0.0 \leq p_i \leq 1.0$

We now illustrate how this vector works by explaining some of its components. The first component of the personality vector represents the *Inclination Level* of *Perceiving* (we symbolize it as *perceivingLevel* for convenience). The lower the value of this component is, the more absentminded the agent. The higher the value of this component is, the more alert the agent. The second component represents the *Focus Aspect* of *Perceiving* (we symbolize it as *perceivingFocus*). When the value of this component is low, the agent focuses more on expectation (the agent is more imaginative). The agent focuses more on reality (the agent is more realistic) when the value of this component is high. Other components of the personality vector are interpreted and symbolized in the same way.

The values of some components of the personality vector are used to modify emotion intensities by influencing the values of the emotion variables. Several components of the personality vector are used to decide the learning rate and how the agent displays emotions.

Table 1. Dimensions of a personality (from [10])

Process	Inclination Level	Illustrated word(s)	Focus Aspect	Illustrated word(s)
Perceiving	Low	Absentminded	Expectations	Imaginative
	High	Alert	Reality	Realistic
Reasoning	Low	Silly	Undesirable effects	Pessimistic
	High	Rational	Desirable effects	Optimistic
Learning	Low	Incurious	What is learned only	Gullible
	High	Curious	What is known only	Intolerant
Deciding	Low	Insecure	First reaction	Impulsive
	High	Self-confident	Good decision	Thoughtful
Acting	Low	Passive	Anything besides the task	Indifferent
	High	Zealous	Result of the task	Perfectionist
Interacting	Low	Introverted	Addressee as a threat	Hostile
	High	Extroverted	Addressee as a help	Friendly
Revealing	Low	Secretive	Lie	Dishonest
	High	Open	Truth	Honest
Feeling emotions	Low	Emotionless	Self	Selfish
	High	Sensitive	Others	Unselfish

5.2 How Personality Affects Emotion in This Model

We now present formulas that show how personality influences the way an agent experiences emotion.

If the agent focuses more on expectations when perceiving (the value of *perceivingFocus* is close to 0.0), then the value of the expectation variables ($P(goal)$ and $P(event)$) have higher influences on emotion intensities:

$$newExpectation = Expectation^{(perceivingFocus+0.5)}$$

If the agent focuses more on undesirable effects (the value of *reasoningFocus* is close to 0.0), the impact variable when negative will have more influence on the intensities of negative emotions. If the agent focuses more on desirable effects, then the impact variable when positive will have more influence on the intensities of positive emotions:

$$newImpact = \begin{cases} -\mid impact \mid^{(reasoningFocus+0.5)} & \text{if } impact < 0 \\ impact^{(1.5-reasoningFocus)} & \text{if } impact \geq 0 \end{cases}$$

The agent with a lower inclination level of feeling emotions (less sensitive) will have lower intensity of emotions for the same event compared to the agent with a higher inclination level of feeling emotions (more sensitive):

$$newImpulse = impulse^{(1.5-feelingLevel)}$$

The agent that focuses more on others when feelings emotions (the value of *feelingFocus* is closed to 0.0) will have higher intensities for **Desire-other** emotions than the agent who focuses more on self when feelings emotions:

$$new\ Desire\text{-}other\ impulse = Desire\text{-}other\ impulse \times feelingFocus$$

The value of *revealingLevel* is used in the emotion displaying component and the value of *learningFocus* influences the learning rate.

6 Learning Components

There are two learning components in the system to make the agent more adaptive. The agent gradually learns the probability of an action's outcome and the values of actions (standard of behavior).

The first component is used to learn the probability of outcomes of an action. Initially, we assign some pre-determined values for the probability of each outcome. We also assign a presumed value for the number of previous observations. This number of previous observations is determined by the learning rate. Thus, the higher the value of this number of counts, the longer it takes to learn new probabilities. We then update these probabilities each time the action is performed through new observation of the actual outcome. Suppose probabilities of outcomes of an action are P_1,P_2,...,P_m. The current number of observations is n. The actual outcome of the action this time is i. Then the probabilities are updated as follows:

$$new\ P_j = \begin{cases} \frac{P_j \times n+1}{n+1} & \text{if } j = i \\ \frac{P_j \times n}{n+1} & \text{if } j \neq i \end{cases}$$

$$new\ n = n + 1$$

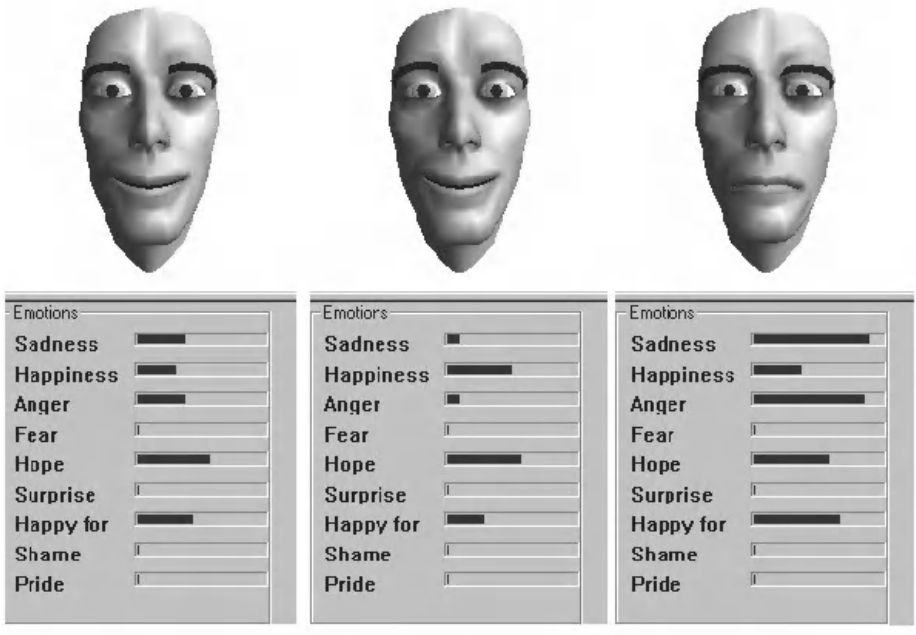


Fig. 2. Neutral Obie (left) expresses facial expressions compared to optimistic Obie (middle) and sensitive Obie (right) after the user eats his bread

The second component is to learn values of actions. For each action that the agent performed, the user can choose to feedback with a point from -1.0 (very bad feedback) to 1.0 (very good feedback). The average point over time is used as the value of the action.

7 Illustration

We have implemented ParleE into Obie, a conversational agent to illustrate the model (see Fig. 2). Obie has been developed with a mixture of text-based and graphical environment to provide interactions with humans. The graphical environment provides a representation of Obie's 3D face that displays emotional facial expressions. We have used OpenGL to implement this 3D face [2]. Obie communicates with the user through text-based interactions.

We have picked a simple domain for this illustration. Obie and the user live in a house and share a car. Whenever Obie is hungry, a goal of feeding himself with the food is initiated. Several actions are available for Obie to achieve the goal: "go shopping", "buy food" and "unload the food from the car". The user can help Obie or can perform some other actions that may prevent Obie from achieving his goal, such as "drive the

car to work". Obie knows the user's goals by reading them from a data file (this process should be done via the dialogue). A piece of dialogue between Obie and the user looks like this:

```
FACT: (in-car bread car1) (at-home car1)
    Choose an action: (-1 to exit)
        0: No Action
        1: (unload-food bread car1)
        2: (go-shopping car1)
        3: (go-working car1)
    You choose: 0
Agent did: (unload-food bread car1)
    case 1 (with probability of 100 %) happened; effects: (in-fridge bread)
FACT: (at-home car1) (in-fridge bread)
```

During the interactions, Obie's emotions are displayed in the 3D face (see Fig. 2). This shows how Obie is capable of expressing his emotions in response to an event.

Figure 2 shows three Obies with different personalities: a neutral, an optimistic and a sensitive one. The scenario was as follows: "Obie goes to the shop and buys bread. He brings the bread home. The user eats his bread." The neutral Obie gets angry after the user eats his bread. As the optimistic Obie tends to ignore this negative event, he is still happy with what have happened before. The sensitive Obie gets very angry with the user. This shows that Obies with different personalities respond differently to an event.

Although we focus on using ParleE for an agent's emotional expressions, ParleE can also be used to alter the agent's behavior. Moreover, with a generic way of appraising events, ParleE can be used in other domains for various application areas.

8 Conclusion

In this paper we have described ParleE, an emotion model for a conversational agent. ParleE enables the conversational agent to respond to the environment and the user with emotional expressions. It is a quantitative, flexible and adaptive model of emotions in which appraising events is based on learning and a probabilistic planning algorithm. ParleE also models personality and motivational states and their roles in determining the way the agent experiences emotion. In the future we will extend the model and test it out for different situations.

References

1. Blum,A. L. and J. C. Langford (1998). Probabilistic planning in the graphplan framework. In *AIPS98 Workshop on Planning as Combinatorial Search*, pages 8–12, June 1998.
2. Bui,T.D., Heylen,D., Poel, M. and A., Nijholt(2001). Generation of facial expressions from emotion using a fuzzy rule based system. In: Proceedings 14th *Australian Joint Conference on Artificial Intelligence (AI 2001)*, December 2001, Adelaide, Australia, Lecture Notes in Artificial Intelligence 2256, M. Stumptner, D. Corbett & M. Brooks (eds.), Springer, Berlin, ISBN 3-540-42960-3, 83–94.

3. Elliott, C. (1992). *The Affective Reasoner: A Process model of emotions in a multi-agent system*. Institute for the Learning Sciences, Evanston, IL: Northwestern University, Ph.D. Thesis.
4. El-Nasr, M. S., Ioerger, T. and J. Yen (1999). FLAME: Fuzzy Logic Adaptive Model of Emotions, *Autonmous Agents and Multi-Agent Systems*, Kluwer Academic Publishers.
5. Fikes, R.E. & N.J. Nilsson (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3–4).
6. Gratch, J. (2000). Émile: Marshalling Passions in Training and Education. In *Proceedings of the 4th International Conference on Autonomous Agents*, Barcelona, Spain, June 2000.
7. Nijholt, A., M. van den Berk and A. van Hessen (1998), A natural language web-based dialogue system with a talking face, *Proceedings Text, Speech and Dialogue*. Sojka et al (eds.), Brno, Czech republic, pp. 415–420.
8. Nijholt, A., D. Heylen and R. Vertegaal (2000), Inhabited interfaces: Attentive conversational agents that help. In: Proceedings 3rd international Conference on Disability, Virtual Reality and Associated Technologies – CDVRAT2000, Alghero, Sardinia.
9. Ortony, A., G.L. Clore, and A. Collins (1988), *The cognitive structure of emotions*, Cambridge, Cambridge University Press.
10. Rousseau, D. (1996). Personality in Computer Characters.In: *Proceedings of the 1996 AAAI Workshop on Entertainment and AI / A-Life*, AAAI Press, Portland, Oregon, August 1996, pp. 38–43.
11. Velásquez, J. (1997). Modeling emotions and other motivations in synthetic agents. In: *Proc. AAAI Conf. 1997*, Providence, RI, pp. 10–15.

Integrating Function Application in State-Based Planning

Ute Schmid[1], Marina Müller[2], and Fritz Wysotzki[3]

[1] Department of Mathematics and Computer Science, University Osnabrück
Albrechtstr. 28, D-49076 Osnabrück, Germany
schmid@informatik.uni-osnabrueck.de
[2] Xtradyne Technologies AG
Schulenburgring 126, D-12102 Berlin, Germany
marinam@cs.tu-berlin.de
[3] Department of Computer Science, Technical University Berlin
Franklinstr. 28, D-10587 Berlin, Germany
wysotzki@cs.tu-berlin.de

Abstract. We present an extension of state-based planning from traditional Strips to function application, allowing to express operator effects as updates. As proposed in PDDL, fluent variables are introduced and, consequently, predicates are defined over general terms. Preconditions of operators are characterized as variable binding constraints with standard preconditions as a special case of equality constraints. Operator effects can be expressed by ADD/DEL effects and additionally by updates of fluent variables. Mixing ADD/DEL effects and updates in an operator is allowed. Updating can involve the application of user-defined and built-in functions of the language in which the planner is realized. We present an operational semantics of the extended language. We will give a variety of example domains which can be dealt with in an uniform way: planning with resource variables, numerical problems such as water jug, functional variants of Tower of Hanoi and blocks-world, list sorting, and constraint-logic programming.

1 Introduction

The development of efficient planning algorithms in the nineties – such as Graphplan (Blum & Furst, 1997), SAT planning (Kautz & Selman, 1996), and heuristic planning (Bonet & Geffner, 1999) – made it possible to apply planning to more demanding real-world domains. Examples are the logistics or the elevator domain (Bacchus et. al., 2000). Many realistic domains involve manipulation of numerical objects. For example, when planning (optimal) routes for delivering objects as in the logistics domain, it might be necessary to take into account time and fuel as resource constraints; plan construction for landing a space-vehicle involves calculations for the correct adjustments of thrusts (Pednault, 1987). One obvious way to deal with numerical objects is to assign and update their values by means of function applications.

While function application is an integral part of deductive planning formalisms (Bibel, 1998; Manna & Waldinger, 1987) this is not true for state-based planning in the Strips tradition. However, the need to extend Strips to obtain more expressive power

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 144–160, 2002.

is recognized in the state-based planning community: The planning domain definition language PDDL (McDermott, 1998) incorporates most features of the action description language ADL (Pednault, 1987). In addition to ADD/DEL effects, ADL allows variable updates by assigning new values which can be calculated by arbitrary functions. E.g., the *put(?x, ?y)* operator in a blocks-world domain might be extended to updating a state variable, such as *?LastBlockMoved := ?x*; the amount of water in a jug $?j_1$ might be changed by a *pour*($?j_1$, $?j_2$) action into $?j_1$:= *max*[0, ($?j_2$ − $?c_1$) + $?j_1$)] where $?j_1$, $?j_2$ are the current quantities of water and $?c_1$ is the capacity of jug $?j_1$ (Pednault, 1994).

In the nineties, most work addressing the semantics of ADL extensions to Strips was concerned with conditional effects and quantification (Koehler, Nebel, & Hoffmann, 1997). Some planners, such as UCPOP, allowed preconditions to include variable binding constraints and interpreted predicates, that is, predicates calling Lisp code, for domains involving arithmetic (Weld, 1994). Updating of fluent variables was only dealt with in a restricted way for specially marked resource variables (Koehler, 1998; Laborie & Ghallab, 1995). For example, the value of a global resource variable *$gas* might be decreased in relation to the distance an airplane flies: *$gas –= distance(?x ?y)/3* (Koehler, 1998). Only recently, Geffner (2000) pointed out that function application in state-based planning is dealt with in a largely ad-hoc manner. He proposed a functional variant of Strips where relations are completely replaced by functional expressions and argued that allowing function symbols in domain modeling results in more efficient encodings. Geffner proposed to introduce functions rather than relations as first-class objects in a domain representation language and presented alternative representations of classic domains, such as Tower of Hanoi, using symbol-manipulating functions. In contrast, within PDDL the focus is on updates of numerical values.

There exist two syntactic variants for modeling updates of numerical values in PDDL (see Fox & Long, 2001): In one variant (McDermott, 1998), fluent variables are declared whose values can be changed, in the second variant (McDermott, 2000), functors are declared as special expressions available for updates. Recently, PDDL 2.1 (Fox & Long, 2001) was presented, which is especially concerned with the semantics of updating numerical variables and durative actions. In PDDL 2.1, the "level 2 semantics" for numerical updates follows the functor proposal of McDermott (2000). Our own work can be seen as an extension of the fluent variables concept of PDDL. Because we are working in the area of combining planning and program synthesis (Schmid & Wysotzki, 2000), we are interested in planning for domains involving function application, such as sorting lists using a *swap*-operator. Furthermore, similar to Geffner (2000), we are not only interested in updating numeric variables but also in a different style of modeling classical domains allowing the use of symbol-manipulating functions.

In the following, we first introduce our functional approach with an example and give several arguments why function application is worth to be considered in state-based planning. Then, we give the extended semantics for state-based planning with functions and discuss plan construction. We present different examples, covering planning with resource constraints, planning with numerical values (water jug problem), a standard programming problem (list sorting), and constraint-logic programming problems. We conclude with a short evaluation and further work to be done.

2 Functional Tower of Hanoi

Introducing functions into planning not only makes it possible to deal with numerical values in a more general way than allowed for by a purely relational language but has several additional advantages which we will illustrate with the Tower of Hanoi domain (see Fig. 1). Similar arguments for introducing functions were given by Geffner (2000).

Allowing not only numerical but also symbol-manipulating functions makes it possible to model operators in a more compact and sometimes also more natural way. For example, representing which disks are lying on which peg in the Tower of Hanoi domain could be realized by a predicate *on(?disks, ?peg)* where $?disks$ represents the ordered sequence of disks on peg $?peg$ with list $[\infty]$ representing an empty peg. Instead of modeling a state change by adding and deleting literals from the current state, arguments of the predicates can be changed by applying standard list-manipulation functions (e. g., built-in Lisp functions like `car(l)` for returning the first element of a list, `cdr(l)` for returning a list without its first element, or `cons(x, l)` for inserting a symbol x as head of a list l).

A further advantage is that objects can be referred to in an indirect way. In the *hanoi* example, `car(l)` refers to the object which is currently the uppermost disk on a peg. There is no need for any additional fluent predicate besides *on(?disks, ?peg)*. The *clear(?disk)* predicate given in the standard definition becomes superfluous. Geffner (2000) points out that indirect reference reduces substantially the number of possible ground atoms and in consequence the number of possible actions, thus plan construction becomes more efficient. Indirect object reference additionally allows for modeling infinite domains while the state representations remain small and compact. For example, `car(l)` gives us the top disk of a peg regardless of how many disks are involved in the planning problem.

(a) Standard representation

Operator: move(?d, ?from, ?to)
PRE: {on(?d, ?from), clear(?d), clear(?to), smaller(?d, ?to)}
ADD: {on(?d, ?to), clear(?from)}
DEL: {on(?d, ?from), clear(?to)}
Goal: {on(d_3, p_3), on(d_2, d_3), on(d_1, d_2)}
Initial State: {on(d_3, p_1), on(d_2, d_3), on(d_1, d_2),clear(d_1), clear(p_2), clear(p_3), smaller(d_1, p_1), smaller(d_1, p_2), smaller(d_1, p_3), smaller(d_2, p_1), smaller(d_2, p_2), smaller(d_2, p_3), smaller(d_3, p_1), smaller(d_3, p_2), smaller(d_3, p_3), smaller(d_1, d_2), smaller(d_1, d_3), smaller(d_2, d_3)}

(b) Functional representation

Operator: move($?p_i$, $?p_j$)
PRE: {on($?l_i$, $?p_i$), on($?l_j$, $?p_j$), car($?l_i$) $/=\infty$, car($?l_i$) $<$ car($?l_j$)}
UPDATE: *change* $?l_j$ *in* on($?l_j$, $?p_j$) *to* cons(car($?l_i$),$?l_j$)
change $?l_i$ *in* on($?l_i$, $?p_i$) *to* cdr($?l_i$)
Goal: {on([∞], p_1), on([∞], p_2), on([1 2 3 ∞], p_3)}
Initial State: {on([1 2 3 ∞], p_1), on([∞], p_2), on([∞], p_3)} ;
∞ represents a dummy bottom disk

Fig. 1. Tower of Hanoi (a) without and (b) with function application

Finally, introducing functions often makes it possible to get rid of static predicates. For example, the *smaller(?x, ?y)* predicate in the *hanoi* domain can be eliminated. By representing disks as numbers, the built-in predicate "$<$" can be used to check whether the application constraint for the *move* operator is satisfied in the current state. Allowing arbitrary boolean operators to express preconditions generalizes matching of literals to contraint satisfaction.

As shown in the *hanoi* example, our approach allows a combination of relational and functional expressions. We extended the Strips planning language to a subset of first order logic, where relational symbols are defined over *terms* with terms as variables, constant symbols, *and* function symbols. Our state representations are still sets of literals, but we introduce a second class of relational symbols (called *constraints*) which are defined over arbitrary functional expressions, as shown in Fig. 1. Standard representations containing only literals defined over constant symbols are included as special case, and we can combine ADD/DEL effects and updates in operator definitions.

3 Extending Strips to Function Applications

In the following we introduce an extension of the Strips language from propositional representations to a larger subset of first order logic, allowing general terms as arguments of relational symbols. Operators are defined over these more general formulas, resulting in a more complex state transition function. The proposed language is compatible with the fluent variable concept of PDDL. The following definition of FPlan is a direct extension of the Strips language. The extensions are given in bold font:

Definition 1 (FPlan Language) *The language $\mathcal{L}(\mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R}_\mathcal{P} \cup \mathcal{R}_\mathcal{C})$ is defined over sets of variables $\mathcal{X}$, constant symbols $\mathcal{C}$,* **function symbols** *$\mathcal{F}$, and of relational symbols $\mathcal{R} = \mathcal{R}_\mathcal{P} \cup \mathcal{R}_\mathcal{C}$ in the following way:*

- *Variables $x \in \mathcal{X}$ are terms.*
- *Constant symbols $c \in \mathcal{C}$ are terms.*
- **If $f \in \mathcal{F}$ is a function symbol with arity $\alpha(f) = i$ and $t_1, t_2, \ldots t_i$ are terms, then $f(t_1, t_2, \ldots, t_i)$ is a term.**
- *There are no other terms.*
- *If $p \in \mathcal{R}_\mathcal{P}$ is a relational symbol with arity $\alpha(p) = j$ and $t_1, t_2, \ldots t_j$ are terms then $p(t_1, t_2, \ldots, t_j)$ is a formula.*
  If $r \in \mathcal{R}_\mathcal{C}$ is a relational symbol with arity $\alpha(r) = j$ and $t_1, t_2, \ldots t_j$ are terms then $r(t_1, t_2, \ldots, t_j)$ is a formula. *We call formulas with relational symbols from $\mathcal{R}_\mathcal{C}$* constraints. *For short, we write $p(\boldsymbol{t})$.*
- *If $p_1(\boldsymbol{t_1})$, $p_2(\boldsymbol{t_2})$, …, $p_k(\boldsymbol{t_k})$ with $p_1, p_2, \ldots, p_k \in \mathcal{R}$, then $\{p_1(\boldsymbol{t_1}), p_2(\boldsymbol{t_2}), \ldots, p_k(\boldsymbol{t_k})\}$ is a formula, representing the conjunction of literals.*
- *There are no other FPlan formulas.*

Remarks:

- *Formulas consisting of a single relational symbol are called (positive) literals.*
  Terms over $\mathcal{C} \cup \mathcal{F}$, i. e., terms without variables, are called ground terms.
  Formulas over ground terms are called ground formulas.
- *With $\mathcal{X}(F)$ we denote the variables occurring in formula F.*

An example of a state representation, goal definition, and operator definition using the FPlan language is given in Fig. 1b for the Tower of Hanoi domain. FPlan is instantiated in the following way: $\mathcal{X} = \{?p_i, ?p_j, ?l_i, ?l_j\}$, $\mathcal{C} = \{p_1, p_2, p_3, 1, 2, 3, \infty\}$, $\mathcal{F} = \{[\,]^i, car^1, cdr^1, cons^2\}$, $\mathcal{R_P} = \{on^2\}$, $\mathcal{R_C} = \{\neq^2, <^2\}$.

A state in the *hanoi* domain, such as $\{$*on([2 3* ∞*], p_1), on([*∞*], p_2), on([1* ∞*], p_3)*$\}$, is given as a set of atoms where the arguments can be arbitrary ground terms. We can use constructor functions to define complex data structures. For example, *[2 3* ∞*]* is a list of constant symbols, where *[ ]* is a list constructor function defined over an arbitrary number of elements. The atom *on([2 3* ∞*], p_1)* corresponds to the evaluated expression *on(cdr([1 2 3* ∞*], p_1))*, where *cdr(?l)* returns a list without its first element. State transformation by updating is defined by such evaluations.

Relational symbols are divided into two categories – symbols in $\mathcal{R_P}$ and symbols in $\mathcal{R_C}$. Symbols in $\mathcal{R_P}$ denote relations which characterize a problem state. For example, *on([1 2 3* ∞*], p_1)* with *on* $\in R_P$ is true in a state, where disks 1, 2, and 3 are lying on peg p_1. Symbols in $\mathcal{R_C}$ denote additional characteristics of a state, namely variable binding constraints. These constraints must not be given explicitely in a state representation but can be checked for a ground formula over $\mathcal{R_P}$. For example, *car(L_1) / $\Leftarrow$* with */* $\Leftarrow \mathcal{R_C}$ is true for L_1 = *[1 2 3* ∞*]* which might be an instantiation of variable $?l_i$ in $on(?l_i, ?p_i)$.

Preconditions are defined as arbitrary formulas over $\mathcal{R_P} \cup \mathcal{R_C}$. Standard preconditions, such as *p = on(*$?L_1$*,* $?x$*)*, can be transformed into constraints: If *match(p,s)* results in $\sigma_1 = \{L_1 \leftarrow [1\ 2\ 3\ \infty], ?x \leftarrow p_1\}$, then the constraints $eq(?x, p_1)$ and $eq(?L_1, [1\ 2\ 3\ \infty])$, with eq as equality-test for symbols in $\mathcal{R_C}$, must hold.

We allow constraints to be defined over free variables, i. e., variables that cannot be instantiated by matching an operator precondition with a current state. For example, variables $?i$ and $?j$ in the constraint $nth(?i, ?L) < nth(?j, ?L)$ might be free. To restrict the possible instantiations of such variables, they must be declared together with a range in the problem specification (an example is given in a later section).

Definition 2 (Free Variables in $\mathcal{R_C}$ and Range) *For a formula $F \in \mathcal{L}(\mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R_P} \cup \mathcal{R_C})$, variables occurring only in literals over $\mathcal{R_C}$ are called free. The set of such variables is denoted as $\mathcal{X_C} \subseteq \mathcal{X}$ and with $\mathcal{X_C}(F)$ we refer to free variables in F. For all variables x in $\mathcal{X_C}$ instantiations must be restricted by a range $R(x)$. For variables belonging to an ordered data type (e. g., natural numbers), a range is declared as $R(x) = [min, max]$ with* min *giving the smallest and* max *the largest value x is allowed to assume. Alternatively, for categorial data types, a range is defined by enumerating all possible values the variable can assume: $R(x) = [v_1, \ldots, v_n]$.*

Definition 3 (State Representation) *A problem state s is a conjunction of atoms over relational symbols in $\mathcal{R_P}$. That is, $s \in \mathcal{L}(\mathcal{C}, \mathcal{F}, \mathcal{R_P})$.*

We presuppose that terms are always *evaluated* if they are grounded. Evaluation of terms is defined in the usual way (Field & Harrison, 1988; Ehrig & Mahr, 1985):

Definition 4 (Evaluation of Ground Terms) *A term t over $\mathcal{L}(\mathcal{C}, \mathcal{F})$ is evaluated in the following way:*

- *eval(c) = c, for $c \in \mathcal{C}$*
- *eval($f(t_1, \ldots, t_n)$) = apply(f(eval(t_1), ..., eval(t_n))), for $f \in \mathcal{F}$ and $t_1, \ldots, t_n \in \mathcal{C} \cup \mathcal{F}$.*

Function application returns a unique value for $f(c_1, \ldots, c_n)$. For constructor functions (such as the list constructor [ ]) we define apply($f_c(c_1, \ldots, c_n)$) = $f_c(c_1, \ldots, c_n)$.

For example, *eval(cdr([1 2 3 ∞]))* returns *[2 3 ∞]*. Evaluation of *eval(plus(3,minus(5,1)))* returns 7. Evaluated states are sets of relational symbols over constant symbols and complex structures, represented by constructor functions over constant symbols. For example, the relational symbol *on* in state description {*on([2 3 ∞], p_1), on([∞], p_2), on([1 ∞], p_3)*} has constant arguments p_1, p_2, and p_3, and complex arguments *[2 3 ∞]*, *[∞]*, and *[1 ∞]*. Relational symbols in $\mathcal{R}_\mathcal{C}$ can be considered as special terms, namely terms that evaluate to a truth value, also called boolean terms. In the following, we will speak of evaluation of expressions when referring to terms including such boolean terms.

Definition 5 (Evaluation of Expressions over Ground Terms)
An expression over $\mathcal{L}(\mathcal{C}, \mathcal{F}, \mathcal{R}_\mathcal{C})$ is evaluated in the following way:

- *eval(c), eval(f) as in Def. 4.*
- *eval($r(t_1, \ldots, t_n)$) = apply(r(eval(t_1), ..., eval(t_n))), for $r \in \mathcal{R}_\mathcal{C}$ and $t_1, \ldots, t_n \in \mathcal{C} \cup \mathcal{F} \cup \mathcal{R}_\mathcal{C}$ with eval($r(t_1, \ldots, t_n)$) $\in \{TRUE, FALSE\}$.*

For a set of atoms over $\mathcal{L}(\mathcal{C}, \mathcal{F}, \mathcal{R}_\mathcal{C})$, i. e., a conjunction of boolean expressions, evaluation is extended to:

$$eval(\{r_1(\boldsymbol{t_1}), \ldots, r_n(\boldsymbol{t_n})\}) = \begin{cases} TRUE & if\ r_i(\boldsymbol{t_i}) = TRUE \\ & for\ i = 1 \ldots n \\ FALSE & else. \end{cases}$$

For example, {*car([1 2 3 ∞]) / car([1 2 3 ∞]) < car([∞])*} evaluates to TRUE. In our planner, evaluation of expressions is realized by calling the meta-function *eval* provided by Common Lisp.

Before introducing goals and operators defined over $\mathcal{L}(\mathcal{C}, \mathcal{F}, \mathcal{R})$, we introduce matching of formulas containing variables (also called patterns) with ground formulas and the extension for constraints. Substitution and matching for the Strips subset $\mathcal{L}_\mathcal{S}$ of FPlan is defined in the usual way:

Definition 6 (Substitution and Matching for Strips) *A substitution is a set of mappings $\sigma = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ defining replacements of variables x_i by terms t_i. For $\mathcal{L}_\mathcal{S}$ terms t_i are restricted to variables and constants. By applying a substitution σ to a formula F – denoted F_σ – all variables $x_i \in \mathcal{X}(F)$ with $x_i \leftarrow t_i \in \sigma$ are replaced by the associated t_i. Note that this replacement is unambiguous, i. e., identical variables are replaced by identical terms. We call F_σ an instantiated formula if all $\mathcal{X}(F)$ are replaced by constant symbols. For a formula $F \in \mathcal{L}_\mathcal{S}$ and a set of atoms $A \in \mathcal{L}_\mathcal{S}(\mathcal{C}, \mathcal{R})$,* match(F,A) = *$\Sigma$ gives all substitutions $\sigma_i \in \Sigma$ with $F_{\sigma_i} \subseteq A$.*

Consider the state

$$A = \{on(d_3, p_1), on(d_2, d_3), on(d_1, p_3), clear(d_1), clear(d_2), clear(p_2), \\ smaller(d_1, p_1), smaller(d_2, p_1), \ldots\}.$$

The formula PRE of the *move* operator given in Fig. 1a

$$F = \{on(?d, ?from), clear(?d), smaller(?d, ?to), clear(?to)\}$$

can be instantiated to $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ with $\sigma_1 = \{?d \leftarrow d_2, ?from \leftarrow d_3, ?to \leftarrow p_2\}$, $\sigma_2 = \{?d \leftarrow d_1, ?from \leftarrow p_3, ?to \leftarrow p_2\}$, $\sigma_3 = \{?d \leftarrow d_1, ?from \leftarrow p_3, ?to \leftarrow d_2\}$.

Definition 7 (Product of Substitutions) *The composition of substitutions $\sigma \circ \sigma'$ is the set of all compatible pairs $(x \leftarrow t) \in \sigma$ and $(x' \leftarrow t') \in \sigma'$ with $x, x' \in \mathcal{X}$ and $t, t' \in \mathcal{L}(\mathcal{X}, \mathcal{C}, \mathcal{F})$. Substitutions are compatible iff for each $(x \leftarrow t) \in \sigma$ there does not exist any $(x' \leftarrow t') \in \sigma'$ with $x = x'$ and $t \not= t'$.*
The product of sets of substitutions $\Sigma = \{\sigma_1, \ldots \sigma_n\}$ and $\Sigma' = \{\sigma'_1, \ldots \sigma'_m\}$ is defined as pairwise composition $\Sigma \cup \Sigma' = \sigma_i \circ \sigma'_j$ for $i = 1 \ldots n$, $j = 1 \ldots m$.

Definition 8 (Matching and Assignment) *Let $F = F_P \cup F_C$ be a formula with*
- *$F_P = \{p_1(\boldsymbol{t_1}), \ldots, p_i(\boldsymbol{t_i})\} \in \mathcal{L}(\mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R}_\mathcal{P})$ and*
- *$F_C = \{r_1(\boldsymbol{t'_1}), \ldots, r_j(\boldsymbol{t'_j})\} \in \mathcal{L}(\mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R}_\mathcal{C})$, and let*
- *$A \in \mathcal{L}(\mathcal{C}, \mathcal{F}, \mathcal{R}_\mathcal{P})$ be a set of evaluated atoms.*

The set of substitutions Σ for F with respect to A is defined as $\Sigma_P \cup \Sigma_C$ with $F_{P_{\sigma_i}} \subseteq A$ and eval($F_{C_{\sigma_i}}$) = TRUE *for all $\sigma_i \in \Sigma$.*
- *Σ_P is calculated by* match(F_P, A) *as specified in Definition 6.*
- *Σ_C is calculated by* ass($\mathcal{X}_\mathcal{C}$) *for all free variables $x_1, \ldots, x_n = \mathcal{X}_\mathcal{C}(F_C)$ such that for all $\sigma_i \in \Sigma_C$ holds $\sigma_i = \{x_1 \leftarrow c_1, \ldots, x_n \leftarrow c_n\}$ where $c_i \in R(x_i)$ (constants c_i in the range of variable x_i).*

Definition 8 extends matching of formulas with sets of atoms in such a way that only those matches are allowed where substitutions additionally fulfill the constraints.

Consider the state

$$A = \{on([2\ 3\ \infty], p_1), on([\infty], p_2), on([1\ \infty], p_3)\}.$$

Formula PRE of the *move* operator given in Fig. 1b

$$F = \{on(?l_i, ?p_i), on(?l_j, ?p_j), car(?l_i)\ / \not= \infty, car(?l_i) < car(?l_j)\}$$

can be instantiated to $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ with

$$\sigma_1 = \{?p_i \leftarrow p_1,\ ?p_j \leftarrow p_2,\ ?l_i \leftarrow [2\ 3\ \infty],\ ?l_j \leftarrow [\infty]\},$$
$$\sigma_2 = \{?p_i \leftarrow p_3,\ ?p_j \leftarrow p_2,\ ?l_i \leftarrow [1\ \infty],\ ?l_j \leftarrow [\infty]\},$$
$$\sigma_3 = \{?p_i \leftarrow p_3,\ ?p_j \leftarrow p_1,\ ?l_i \leftarrow [1\ \infty],\ ?l_j \leftarrow [2\ 3\ \infty]\}$$

but not to

$$\sigma_4 = \{?p_i \leftarrow p_1, ?p_j \leftarrow p_3, ?l_i \leftarrow [2\ 3\ \infty], ?l_j \leftarrow [1\ \infty]\},$$
$$\sigma_5 = \{?p_i \leftarrow p_2, ?p_j \leftarrow p_1, ?l_i \leftarrow [\infty], ?l_j \leftarrow [2\ 3\ \infty]\},$$
$$\sigma_6 = \{?p_i \leftarrow p_2, ?p_j \leftarrow p_3, ?l_i \leftarrow [\infty], ?l_j \leftarrow [1\ \infty]\}.$$

A procedural realization for calculating all instantiations of a formula with respect to a set of atoms is to first calculate all matches and then modify Σ by stepwise introducing the constraints. For the example above, we first calculate $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6\}$ considering only the sub-formula $F_P = \{on(?p_i, ?l_i), on(?p_j, ?l_j)\}$. Next, constraint $car(?l_i)\ / \not= \infty$ is introduced, reducing Σ to $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$. Finally, we obtain $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ by applying the constraint $car(?l_i) < car(?l_j)$.

If the constraints contain free variables, each substitution in Σ is combined with instantiations of these variables as specified in Definition 7. An example for instantiating formulas with free variables is given the example section for the list sorting domain.

Definition 9 (Goal Representation) *A goal $\mathcal{G}$ is a formula in $\mathcal{L}(\mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R})$.*

The goal given for *hanoi* in Fig. 1b can alternatively be represented by:

$$\{\ \ on([1\ 2\ 3\ \infty], p_3), on(?l_i, ?p_i), on(?l_j, ?p_j), \infty = car(?l_i), \infty = car(?l_j)\}.$$

Definition 10 (FPlan Operator) *An FPlan operator op is described by preconditions (PRE), ADD and DEL lists, and updates (UP) with*
$PRE = PRE_{\mathcal{P}} \cup PRE_{\mathcal{C}} \in \mathcal{L}(\mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R}_{\mathcal{P}} \cup \mathcal{R}_{\mathcal{C}})$ *and* $ADD, DEL \in \mathcal{L}(\mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R}_{\mathcal{P}})$.
An update UP is a list of function applications, specifying that a function $f(\boldsymbol{t}) \in \mathcal{L}(\mathcal{X}, \mathcal{C}, \mathcal{F})$ is applied to a term t'_i which is argument of literal $p(\boldsymbol{t'})$ with $p \in \mathcal{R}_{\mathcal{P}}$.

An example for an update effect is given in Fig. 1b:

change $?l_i$ **in** $on(?l_i, ?p_i)$ **to** $cdr(?l_i)$.

defines that variable $?l_i$ in literal $on(?l_i, ?p_i)$ is updated to $cdr(?l_i)$. If more than one update effect is given in an operator, the updates are performed sequentially from the uppermost to the last update. An update effect cannot contain free variables, that is $\mathcal{X}(UP) \subseteq \mathcal{X}(PRE)$.

An operator is instantiated by matching and assignment with respect to a given state as described above. This corresponds to obtaining grounded actions by normalization as proposed by Fox and Long (2001).

Definition 11 (Updating a State) *For a state $s = \{p_1(\boldsymbol{t_1}), \ldots, p_n(\boldsymbol{t_n})\}$ and an instantiated operator o with update effects $u \in UP$, updating is defined as replacement $t'_i := eval(f(\boldsymbol{t}))$ for a fixed argument t'_i in a fixed literal $p_i(\boldsymbol{t'}) \in s$ with $\mathcal{X}(\boldsymbol{t}), \mathcal{X}(\boldsymbol{t'}) \subseteq \mathcal{X}(PRE)$.*
For short we write update(u, s).
Applying a list of updates $UP = (u_1, \ldots, u_n)$ to a state s is defined as $update(UP, s) = update(u_n, update(u_{n-1}, \ldots, update(u_1, s)) \ldots)$.

For the initial state of *hanoi* in Fig. 1, $s = \{on(p_1, [1\ 2\ 3\ \infty]), on(p_2, [\infty]), on(p_3, [\infty])\}$, the update effects of the *move* operator can be instantiated to

$u_1 =$ **change** $[\infty]$ **in** $on(p_3, [\infty])$ **to** $cons(car([1\ 2\ 3\ \infty])[\infty])$
$u_2 =$ **change** $[1\ 2\ 3\ \infty]$ **in** $on(p_1, [1\ 2\ 3\ \infty])$ **to** $cdr([1\ 2\ 3\ \infty])$.

Applying these updates to s results in:

$$\begin{aligned}
&update((u_1, u_2), \{on(p_1, [1\ 2\ 3\ \infty]), on(p_2, [\infty]), on(p_3, [\infty])\}) \\
&= update(u_2, \{on(p_1, [1\ 2\ 3\ \infty]), on(p_2, [\infty]), on(p_3, eval(cons(car([1\ 2\ 3\ \infty]), [\infty])))\} \\
&= update(u_2, \{on(p_1, [1\ 2\ 3\ \infty]), on(p_2, [\infty]), on(p_3, [1\ \infty])\} \\
&= \{on(p_1, eval(cdr([1\ 2\ 3\ \infty]))), on(p_2, [\infty]), on(p_3, [1\ \infty])\} \\
&= \{on(p_1, [2\ 3\ \infty]), on(p_2, [\infty]), on(p_3, [1\ \infty])\}.
\end{aligned}$$

Note that the operator is fully instantiated before the update effects are calculated. The current substitution $\sigma \in \Sigma$ is *not* affected by updating. That is, if the value of an instantiated variable is changed, as for example $L_2 = [\infty]$ is changed to $L_2 = [1\ \infty]$, this change remains local to the literal specified in the update.

Operator:	move($?p_i$, $?p_j$)
PRE:	{on ($?l_i$, $?p_i$), on($?l_j$, $?p_j$), not-empty($?l_i$), legal($?l_i$, $?l_j$)}
UPDATE:	*change* $?l_j$ *in* on($?l_j$, $?p_j$) *to* cons(car($?l_i$), $?l_j$)
	change $?l_i$ *in* on($?l_i$, $?p_i$) *to* cdr($?l_i$)
Goal:	{on([ ], p_1), on([ ], p_2), on([1 2 3], p_3)}
Initial State:	{on([1 2 3], p_1), on([ ], p_2), on([ ], p_3)}
Functions:	not-empty(l) = not(null(l))
	legal(l_1, l_2) = *if* null(l_2) *then* TRUE *else* $<$ (car(l_1), car(l_2))

Fig. 2. Tower of Hanoi with user-defined functions

Definition 12 (Operator Application) *For a state s and an instantiated operator o, operator application is defined as $Res(o, s) = update(UP, s \setminus DEL(o) \cup ADD(o))$ if $PRE_P(o) \subseteq s$ and $eval(PRE_C(o)) = TRUE$.*

ADD/DEL effects are always calculated *before* updating. Examples for operators with combined ADD/DEL and update effects are given below. As mentioned above, the set of function symbols $\mathcal{F}$ and the set of relational symbols $\mathcal{R}_\mathcal{C}$ of FPlan can contain arbitrary built-in and user-defined functions. When parsing a problem specification, each symbol which is part of Common Lisp and each symbol corresponding to the name of a user-defined function is treated as expression to be evaluated. Symbols in $\mathcal{R}_\mathcal{C}$ have to evaluate to a truth value.

An example specification of *hanoi* using built-in and user-defined functions is given in Fig. 2.[1] For the Tower of Hanoi domain the FPlan language is instantiated to $\mathcal{X} = \{?p_i, ?p_j, ?l_i, ?l_j\}$, $\mathcal{C} = \{p_1, p_2, p_3, 1, 2, 3, nil\}$, $\mathcal{F} = \{[\,]^i, car^1, cdr^1, cons^2\}$, $\mathcal{R}_\mathcal{P} = \{on^2\}$, $\mathcal{R}_\mathcal{C} = \{not\text{-}empty^1, legal^2\}$. The user-defined functions can be defined over additional built-in and user-defined functions which are not included in $\mathcal{L}$. For a Lisp-implemented planner after loading a problem specification, all declared user-defined functions are appended to the set of built-in Lisp functions and interpreted in the usual manner.

An FPlan *planning problem* is given as $P(\mathcal{O}, \mathcal{I}, \mathcal{G})$, where operators $\mathcal{O}$, initial states $\mathcal{I}$, and goals $\mathcal{G}$ are defined over the FPlan language, which extends Strips by allowing function applications. A plan for the *hanoi* problem specified in Fig. 1b is given in Fig. 3. Plan construction can be performed using either forward or backward search in the state space. To make FPlan applicable for backward planners, for each function f appearing in an update-effect, its inverse function must be given, that is, for backwards planning only bijective functions are admissible. Backward planning with function application is described in detail in (Müller, 2000). When allowing arbitrary user-defined functions, termination of plan construction can no longer be guaranteed. This holds especially for user-defined functions involving loops or recursion.

[1] For a Lisp implemented planning system, functions must be defined in Lisp syntax, e. g.,:
`(defun legal (l1 l2) (if (null l2) T (< (car l1) (car l2))))`.

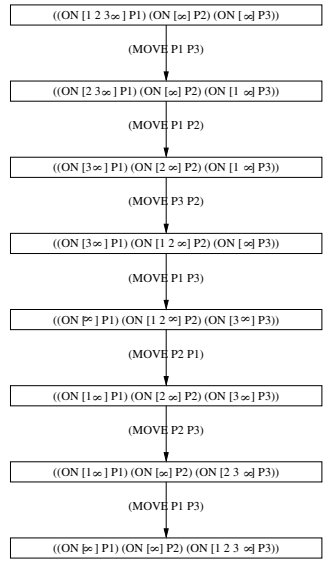


Fig. 3. A plan for Tower of Hanoi

4 Examples

In the previous section, we presented FPlan with a functional version of the Tower of Hanoi domain, demonstrating how operators with ADD/DEL effects can be alternatively modeled with update effects. In the following, we will give a variety of examples for problem specifications with FPLan. First we will show how problems involving resource constraints can be modeled. Afterwards, we will give an example for a numerical domain. We will show how standard programming problems can be modeled in planning, and finally we will present preliminary ideas for combining planning with constraint logic programming.

The algorithm for planning with function-applications follows directly from the definitions given above. The main modification of a standard state-based planner concerns the calculation of a new state from a given state (Def. 12). All examples presented here were tested with our backward planner DPlan which is a kind of universal planner, based on calculating pre-images by breadth-first search (Schmid & Wysotzki, 2000). DPlan was extended for domains presented in the FPlan language and the results are reported in Müller (2000). For this paper, plans were constructed with forward planning, but still using DPlan's breadth-first strategy. Because work on function application in state-based planning is still at its beginning, we were more concerned with feasibility than with efficiency and we focus on how function application can be modeled with FPlan and do not present empirical evaluations of performance.

Operator: **fly**($?plane$, $?x$, $?y$)
PRE: {at($?plane$, $?x$), fuel-res($?plane$, $?fuel$), $?fuel \geq$ distance($?x$, $?y$)/3}
ADD: {at($?plane$, $?y$)}
DEL: {at($?plane$, $?x$)}
UPDATE: *change* $?fuel$ *in* fuel-res($?plane$, $?fuel$) *to* calc-fuel($?fuel$, $?x$, $?y$)
change $?time$ *in* time-res($?plane$, $?time$) *to* calc-time($?time$, $?x$, $?y$)
Goal: {at(p1, berlin)}
Initial State: {at(p1, london), fuel-res(p1, 750), time-res(p1, 0)}
Functions: calc-fuel($?fuel$, $?x$, $?y$) = $?fuel$ − distance($?x$, $?y$)/3
calc-time($?time$, $?x$, $?y$) = $?time$ + 3/20 ∗ distance($?x$, $?y$)

Fig. 4. A Problem specification for the *airplane* domain

Table 1. Database with distances between airports

	Berlin	Paris	London	New York
Berlin	–	540 m	570 m	3960 m
Paris	540 m	–	210 m	3620 m
London	570 m	210 m	–	3460 m
New York	3960 m	3620 m	3460 m	–

4.1 Planning with Resource Variables

Planning with resource constraints can be modeled as a special case of function updates with FPlan. As an example we use an airplane domain (Koehler, 1998). A problem specification in FPlan is given in Fig. 4.

The operator *fly* specifies what happens when an airplane $?plane$ flies from airport $?x$ to airport $?y$. We consider two resources: the amount of fuel and the amount of time the plane needs to fly from one airport to another. In the initial state the tank is filled to capacity (*fuel-res(750)*) and no time has yet been spent (*time-res(0)*). A resource constraint that must hold before the action of flying the plane from one airport $?x$ to another airport $?y$ can be carried out could be *?fuel* $\geq$ *distance(*$?x$, $?y$*)/3*, that is, the plane has to be at airport $?x$ and there has to be enough fuel in the tank to travel the distance from $?x$ to $?y$. For our example, we give no time constraint as precondition for flying, but the time resource is changed by operator application. The precondition could include a time constraint, such as, *time-res(*$?plane$, $?time$*), ?time* $\geq$ *distance(*$?x$, $?y$*)* $\cdot \frac{3}{20}$.

The distance between two airports is obtained by calling the function *distance*(?x, ?y) which returns the distance value by consulting an underlying database (see tab. 1). The distances can alternatively be modeled as static predicates (e. g., *distance(berlin, paris, 540)*, etc.).

The position of the plane $?plane$ is modeled with the literal *at*. When flying from $?x$ to $?y$ the literal *at(*$?plane$, $?x$*)* is deleted from and the literal *at(*$?plane$, $?y$*)* is added to the set of literals describing the current state. Note, that we could have omitted the ADD/DEL effect by modeling the position of the plane as fluent variable. The resource variable $?fuel$ described by the relational symbol *fuel-res* is updated with the result of the user-defined function *calc-fuel* which calculates the consumption of fuel according

to the traveled distance. The resource variable $?time$ described by the relational symbol *time-res* is updated in a similar way assuming that it takes $3/20 * distance(?x, ?y)$ to fly the distance from airport $?x$ to $?y$. When ADD/DEL and update effects occur together in one operator the update effect is carried out on the state obtained *after* calculating the ADD/DEL effect (see Def. 12).

Updating of state variables as proposed in FPlan is more flexible than handling resource variables separately (as in Koehler (1998). While in Koehler (1998), *fuel* and *time* are global variables, in FPlan the current values of *fuel* and *time* are arguments of relational symbols *fuel-res(?plane, ?fuel)*, *time-res(?plane,?fuel)*. Therefore, while modeling a problem involving more than one plane can be easily done in FPlan this is not possible with Koehler's approach. For example we can specify the following goal and initial state:

Goal:	{at(p1, berlin), at(p2, paris)}
Initial State:	{at(p1, berlin), fuel-res(p1, 750), time-res(p1, 0), at(p2, paris), fuel-res(p2, 750), time-res(p2, 0)}

Modeling domains with time or cost resources is simple when function applications are allowed. Typical examples are job scheduling problems – such as the machine shop domain presented in Veloso et al. (1995). To model time steps, a relational symbol *time(?t)* can be introduced. Time $?t$ is initially zero and each operator application results in $?t$ being incremented by one step. To model a machine that is occupied during a certain time interval, a relational symbol *occupied(?m, ?o)* can be used where $?m$ represents the name of a machine and $?o$ the last time slot where it is occupied with $?o = 0$ representing that the machine is free to be used. For each operator involving the usage of a machine, a precondition requesting that the machine is free for the current time slot can be introduced. If a machine is free to be used, the *occupied* relation is updated by adding the current time and the amount of time steps the executed action requests. It can also be modeled that occupation time does not only depend on the kind of action performed but also on the kind of object involved (e. g., polishing a large object could need three time steps, while polishing a small object needs only one time step).

4.2 Planning for Numerical Problems – The Water Jug Domain

Numerical domains as the *water jug* domain presented by Pednault (1994), cannot conveniently be modeled with a Strips-like representation. Figure 5 shows an FPlan speci-

Operator:	**pour**($?j1$, $?j2$)
PRE:	{volume($?j1$, $?v1$), volume($?j2$, $?v2$), capacity($?j2$, $?c2$), ($?v2 < ?c2$), ($?v1 > 0$)}
UPDATE:	*change* $?v1$ *in* volume($?j1$, $?v1$) *to* $max(0,\ ?v1 - ?c2 + ?v2)$
	change $?v2$ *in* volume($?j2$, $?v2$) *to* $min(?c2,\ ?v1 + ?v2)$
Goal:	{volume(a, 25), volume(b, 0), volume(c, 52)}
Statics:	{capacity(a, 36), capacity(b, 45), capacity(c, 54)}
Initial State:	{volume(a, 36), volume(b, 7), volume(c, 34) }

Fig. 5. A problem specification for the *water jug* domain

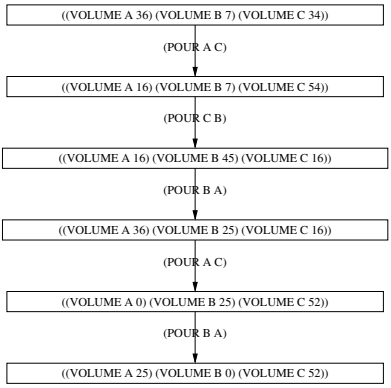


Fig. 6. A plan for a *water jug* problem

fication of the *water jug* domain: We have three jugs of different volumes and different capacities. Operator *pour* models the action of pouring water from one jug *?j1* into another jug *?j2* until either *?j1* is empty or *?j2* is filled to capacity. The capacities of the jugs are statics while the actual volumes are fluents. The resulting volume *?v1* for jug *?j1* is either zero or what remains in the jug when pouring as much water as possible into jug *?j2*: $\max[0, v1 - c2 + v2]$. The resulting volume *?v2* for jug *?j2* is the either its capacity or its previos volume plus the volume of the first jug *?j1*: $\min[c2, v1 + v2]$. The resulting plan for the given problem specification is shown in Fig. 6.

4.3 Planning for Programming Problems – Sorting of Lists

With the extension to functional representation a number of programming problems, for example list sorting algorithms such as bubble, merge or selection sort, can be planned more efficiently. Recently, there is some interest in applying planning to standard programming problems in the context of policy learning (Bonet & Geffner, 1998; Schmid & Wysotzki, 2000). We have specified selection sort in the standard and the functional way (see Fig. 7).

In the functional representation we can use the built-in function "$>$" instead of the predicate *greater*, the constructor for lists *slist*, and the function *nth(n, L)* to reference the n-th element of the list L instead of specifying the position of each element in the list by using the literal *is–at(?x, ?i)*. In the functional representation, the indices to the list *?i* and *?j* are free variables, which – for lists with three elements – can range from zero to two. When calculating the possible instantiations for the operator *swap*, the variables *?i, ?j* can be assigned any value within their range for which the precondition holds (see Def. 2). The function swap is defined as an external function using a function *substitute(p, q, L)* that replaces q with p in a list L.

4.4 Constraint Satisfaction as Special Case of Planning

Logical programs can be defined over rules and facts (Sterling & Shapiro, 1986). Rules correspond to planning operators. Facts are similar to static relations in planning, that is, they are assumed to be true in all situations. Constraint logic programming extends standard logic programming such that constraints can be used to efficiently restrict variable bindings (Frühwirth & Abdennadher, 1997). In Fig. 8 an example for a program in constraint Prolog is given. The problem is to compose a light meal consisting of an appetizer, a main dish and a dessert which all together contains not more than a given calorie value.

The same problem can be specified in FPlan (see Fig. 9). The goal consisting of a conjunction of literals now contains free variables and a set of constraints determining the values of those variables. In this case $?x$, $?y$, $?z$ are free variables that can be assigned any value within their range (here enumerations of dishes). The ranges of the variables $?x$, $?y$, $?z$ are specified for the domain and not for a problem. The function *calories* looks up the calorie value of a dish in an association list.

There are no operators specified for this example. The planning process in this case solves the goal constraint by finding those combinations of dishes that satisfy the

(a) Standard representation

Operator:	**swap**($?i$, $?j$, $?x$)
PRE:	{is–at($?x$, $?i$), is–at($?y$, $?j$), greater($?x$, $?y$)
ADD:	{is–at($?x$, $?j$), is–at($?y$, $?i$)}
DEL:	{is–at($?x$, $?i$), is–at($?y$, $?j$)}
Initial State:	{is–at(3, p1), is–at(2, p2), is–at(1, p3)}
Goal:	{is–at(1, p1), is–at(2, p2), is–at(3, p3)}

(b) Functional representation

Operator:	**swap**($?i$, $?j$, $?x$)
PRE:	{slist($?x$), nth($?i$, $?x$) > nth($?j$, $?x$)}
UPDATE:	*change* $?x$ *in* slist($?x$) *to* swap($?i$, $?j$, $?x$)
Variables:	{($?i$:range (0 2)) ($?j$:range (0 2))}
Initial State:	{slist([3 2 1])}
Goal:	{slist([1 2 3])}
Functions:	swap ($?i$, $?j$, $?x$) =
	if $?i$ >$?j$ then swap($?j$, $?i$, $?x$) else substitute(nth($?i$, $?l$), nth($?j$, $?l$),
	substitute(nth($?j$, $?l$), nth($?i$, $?l$), $?l$:start $?i$:end $?j$) :start $?j$:end $?j + 1$)

Fig. 7. Specification of *selection sort*

```
lightmeal(A, M, D) ← 1400 ≥ I + J + K, I > 0, J > 0, K > 0,
                     appetizer(A, I), main(M, J), dessert(D, K).
appetizer(radishes, 50). main(beef, 1000).  dessert(icecream, 600).
appetizer(soup, 300).    main(pork, 1200). dessert(fruit, 90).
```

Fig. 8. *Lightmeal* in constraint prolog

Domain:	Lightmeal
variables:	($?x$:range (radishes, soup), $?y$:range (beef, pork), $?z$:range (fruit, icecream))
Initial State:	{ } ; empty
Goal:	{(lightmeal($?x$, $?y$, $?z$), (calories($?x$) + calories($?y$) + calories($?z$))$\leq$ 1400}
Functions:	calories($?dish$) = cadr(assoc($?dish$, calorie-list))
	calorie-list := ((radishes 50) (beef 1000) (fruit 90) (icecream 600) (soup 300) (pork 1200) ...)

Fig. 9. *Lightmeal* in FPlan

constraint. For example a meal consisting of radishes with 50 calories, beef with 1000 calories and fruit with 90 calories.

5 Conclusion and Future Work

We have presented an extension of the Strips planning language to function application with the following characteristics: Planning operators can be defined using arbitrary symbolical and numerical functions; ADD/DEL effects and updates can be combined; indirect reference to objects via function application allows for infinite domains; planning with resource constraints can be handled as special case.

The proposed language FPlan can be used to give function application in PDDL (McDermott, 1998) a clear semantics. The described semantics of operator application can be incorporated in arbitrary Strips planning systems. In contrast to other proposals for dealing with the manipulation of state variables, we do not represent them as specially marked "first class objects" (Fox & Long, 2001) but as arguments of relational symbols. This results in a greater flexibility of FPlan, because each argument of a relational symbol may principally be changed by function application. As a consequence, we can model domains usually specified by operators with ADD/DEL effects alternatively by updating state variables (Geffner, 2000).

Furthermore, allowing arbitrary numerical and symbol-manipulating functions makes it possible to apply planning to a larger class of domains. Our planning system DPlan is used as a tool for inductive synthesis of functional programs (Schmid & Wysotzki, 1998, 2000). In this context, we are interested in constructing plans for standard programming problems as list sorting or reversing of lists.

Of course, our proposal is just a first step to integrate function application into state-based planning. Many questions are open to further research. First, it might be interesting to contrast FPlan with PDDL 2.1. Fox and Long (2001) argues against fluent variables because they do not define finite ranges and proposes specially marked functional expressions instead. To overcome this problem, we must introduce an explicit restriction of all free variables in a given problem definition (as in the sorting example) or in the domain (as in the lightmeal example). Furthermore, it would be interesting to investigate how variable updates can interact with other PDDL features, especially with all-quantification. This would be a further step in combining planning with (constraint) logic programming.

Acknowledgements. Thanks to Petra Hofstedt for helpful discussions and to Laurie Hiyakumoto, Janin Toussaint, and Timo Steffens for critical comments on an earlier draft of this paper.

References

Bacchus, F., Kautz, H., Smith, D. E., Long, D., Geffner, H., & Koehler, J. (2000). *AIPS-00 Planning Competition, Breckenridge, CO.*

Bibel, W. (1998). Let's plan it deductively. *Artificial Intelligence*, *103*(1–2), 183–208.

Blum, A., & Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*(1–2), 281-300.

Bonet, B., & Geffner, H. (1998). Learning sorting and decision trees with POMDPs. In *Proc. 15th international conf. on machine learning* (pp. 73–81). Morgan Kaufmann.

Bonet, B., & Geffner, H. (1999). Planning as heuristic search: New results. In *Proc. European Conference on Planning (ECP-99), Durham, UK.* Springer.

Ehrig, H., & Mahr, B. (1985). *Fundamentals of algebraic specification 1.* Springer.

Field, A. J., & Harrison, P. G. (1988). *Functional progamming.* Reading, MA: Addison-Wesley.

Fox, M., & Long, D. (2001). *PDDL2.1: An extension to PDDL for expressing temporal planning domains.*
`http://www.dur.ac.uk/d.p.long/competition.html`.

Frühwirth, T., & Abdennadher, S. (1997). *Constraint-programming.* Berlin: Springer.

Geffner, H. (2000). Functional Strips: A more flexible language for planning and problem solving. In J. Minker (Ed.), *Logic-based artificial intelligence.* Dordrecht: Kluwer.

Kautz, H., & Selman, B. (1996). Pushing the envelope: Planning, propositional logic and stochastic search. In *Proc. 13th national conference on artificial intelligence and 8th innovative applications of artificial intelligence conference* (pp. 1194–1201).

Koehler, J. (1998). Planning under resource constraints. In H. Prade (Ed.), *Proc. 13th European Conference on Artificial Intelligence (ECAI-98* (p. 489–493). Wiley.

Koehler, J., Nebel, B., & Hoffmann, J. (1997). Extending planning graphs to an ADL subset. In *Proc. European Conference on Planning (ECP-97)* (p. 273–285). Springer. (extended version as Technical Report No. 88/1997, University Freiburg)

Laborie, P., & Ghallab, M. (1995). Planning with sharable resource constraints. In *Proc. of the 14th IJCAI* (p. 1643–1649). Morgan Kaufmann.

Manna, Z., & Waldinger, R. (1987). How to clear a block: a theory of plans. *Journal of Automated Reasoning*, *3*(4), 343–378.

McDermott, D. (1998). *PDDL – the planning domain definition language.*
`http://ftp.cs.yale.edu/pub/mcdermott`.

McDermott, D. (2000). The 1998 AI planning systems competition. *AI Magazine*, *21*(2).

Müller, M. (2000). *Integration von Funktionsanwendungen beim zustandsbasierten Planen.* diploma thesis, Dep. of Computer Science, TU Berlin.

Pednault, E. P. D. (1987). Formulating multiagent, dynamic-world problems in the classical planning framework. In M. P. Georgeff & A. L. Lansky (Eds.), *Proc. Workshop on Reasoning About Actions and Plans* (pp. 47–82). Morgan Kaufmann.

Pednault, E. P. D. (1994). ADL and the state-transition model of action. *Journal of Logic and Computation*, *4*(5), 467–512.

Schmid, U., & Wysotzki, F. (1998). Induction of recursive program schemes. In *Proc. 10th European Conference on Machine Learning (ECML-98)* (Vol. 1398, p. 214–225). Springer.

Schmid, U., & Wysotzki, F. (2000). Applying inductive programm synthesis to macro learning. In *Proc. 5th Int.. Conf. on Artificial Intelligence Planning and Scheduling* (p. 371–378).

Sterling, L., & Shapiro, E. (1986). *The art of Prolog: Advanced programming techniques.* MIT Press.

Veloso, M., Carbonell, J., Pérez, M. A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating planning and learning: The Prodigy architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, *7*(1), 81–120.

Weld, D. (1994). An introduction to least commitment planning. *AI Magazine*, *15*(4), 27–61.

Fast Winner-Takes-All Networks for the Maximum Clique Problem

Brijnesh J. Jain and Fritz Wysotzki

Dept. of Computer Science, Technical University Berlin, Germany

Abstract. We present an in-depth mathematical analysis of a winner-takes-all Network tailored to the maximum clique problem, a well-known intractable combinatorial optimization problem which has practical applications in several real world domains. The analysis yields tight bounds for the parameter settings to ensure energy descent to feasible solutions. To verify the theoretical results we employ a fast annealing schedule to the WTA algorithm and show the effectiveness of the proposed approach for large scaled problems in extensive computer simulations.

1 Introduction

A clique of a graph is a fully connected substructure. Cliques with maximal number of vertices are the maximum cliques of a graph. Finding the maximum clique is a classical NP-complete problem in combinatorial optimization [5]. Feige et al. have shown in [3] that even approximating the maximum clique of a graph is almost NP-complete. Therefore exact algorithms which guarantee optimal solutions are useless for all but the smallest input graphs, since the execution times of these algorithms grow exponentially with the number of vertices. On the other hand, in practice it is often required to find a sufficient good approximate solution within a reasonable time.

Several real world applications can be mapped onto the maximum clique problem. Balas and Yu [1] mentioned information retrieval, experimental design, signal transmission, and computer vision. An application of the maximum clique problem for labeled point pattern matching was proposed by Ogawa in [13]. Further applications are PLA folding [12], the stereo vision correspondence problem [9], visual shape recognition [14], cluster analysis, classification, and pattern recognition of structured objects [15,16], graph coloring, disposal systems, VLSI circuit design, and biological systems [6].

Due to its applicability in various fields many heuristics have been devised to approximately solve the maximum clique problem. For details we refer to a survey by Bomze et al. [2]. Among other heuristics artificial neural networks have been successfully employed to find good approximations of a maximum clique in a given graph [2,17]. Following the seminal paper of Hopfield and Tank [8], the general approach to solve combinatorial optimization problems maps the objective function of the optimization problem onto an energy function of a neural network. The constraints of the problem are included in the energy

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 163–173, 2002.

function as penalty terms, such that the global minima of the energy function correspond to the solutions of the combinatorial optimization problem. In order to improve the solution quality many variations of the Hopfield Network have been proposed (for an overview see references in [2,17]). However, in almost all cases of the continuous Hopfield model and its variants the free parameters are chosen heuristically and a theoretical justification is missing.

The best results were reported by neural algorithms and related variants which are also the slowest [10]. In particular, mean-field annealing and simulated annealing yield the best solutions but their computational effort may be prohibitive for large graphs and for repeated application of the maximum clique problem on a large set of graphs as in pattern recognition or clustering tasks of structured objects [11,14,15,16]. For the practical use efficient neural maximum clique solver are required. In general an optimization algorithm is inefficient if it spends lot of computational efforts for less avail in the sense, that the solution does not significantly improve. The significance of an improvement is highly dependent on the desired precision of the results and on the envisaged goals. Each improvement corresponds to an economical profit and each iteration causes costs. From this point of view an optimization can run without loss up to a fixed precision. Improvements obtained without loss are considered to be significant. Following these considerations, simulated annealing, mean-field annealing, and related approaches might possibly not be efficient in some application areas.

Consequently, in this paper we introduce a rapid and simple winner-takes-all (WTA) Network for the maximum clique problem. We rigorously analyze our proposed model and derive bounds for the parameters to ensure that the WTA network performs an energy descent until it terminates in a stable state which correspond to a feasible solution. The theoretical results may give new insight into the functioning of a neural network for combinatorial optimization problems. On the basic of our mathematical analysis we incorporate an annealing scheme to speed up our algorithm. In an empirical study we show, that the WTA algorithm outperforms greedy-like algorithms. Thus the simplicity of the network and the theoretical results yield guidelines for the unskilled practitioner how to employ the WTA network in practical applications without searching for an adequate parameter setting in a laborious trial and error procedure.

The rest of this paper is organized as follows: Section 2 describes the maximum clique problem. Section 3 presents a WTA network for solving the maximum clique problem. In Sect. 4 we give an in-depth analysis of the WTA model. In Sect. 5 we conduct extensive computer simulation to test the performance of the proposed WTA heuristics. The last section concludes this contribution.

2 The Maximum Clique Problem

A *graph* is a pair $G = (V, E)$ consisting of a finite set $V \neq \emptyset$ of *vertices* and a binary relation $E \subseteq [V]^2$ where $[V]^2$ is the set of all 2-element subsets $\{i, j\} \subseteq V$. The elements of E are called *edges*. The *size* $|G|$ of a graph G is the number of its vertices.

A subset C of V is called *clique* if $[C]^2 \subseteq E$. A *maximum clique* is a clique with maximum cardinality of vertices. A *maximal clique* is a clique which is not contained in any larger clique. The *clique number* $\omega(G)$ of a graph G is the number of vertices of a maximum clique in G. The *maximum clique problem* is the problem of finding a maximum clique in a given graph.

3 Clique Detection with WTA Networks

The (continuous) WTA model is composed of a set of fully interconnected processing elements (units). The time-discrete dynamics of the system is of the form

$$x_i(t+1) = x_i(t) + \sum_{j \not= i} w_{ij} h_j^\tau(t) \tag{1}$$

where $x_i(t) \in \mathbb{R}$ denotes the activation of unit i at time t and $w_{ij} = w_{ji}$ is the synaptic weight between unit i and unit j. The output $h_i^\tau(t)$ of unit i is computed by a limiter transfer function of the form

$$h_i^\tau(t) = \begin{cases} 1 & : \quad x_i(t) \geq \tau \\ 0 & : \quad x_i(t) \leq 0 \\ x_i(t)/\tau & : \quad \text{otherwise} \end{cases} \tag{2}$$

where $\tau > 0$ is a control parameter called *pseudo-temperature* or *gain*. Starting with a sufficient large initial value $\tau = \tau_0$ the pseudo-temperature is decreased according to an *annealing-schedule* to a final value $\tau = \tau_f$.

In order to solve the maximum clique problem for a given graph $G = (V, E)$ the network consists of $|V| = n$ units which are connected with weight $w_{ij} = w_E > 0$ if $\{i, j\} \in E$ is an edge in G and with weight $w_{ij} = -w_I < 0$ if $\{i, j\} \notin E$. Self weights w_{ii} are set to zero. In this way a graph G uniquely[1] determines the topology of a neural maximal clique solver. For this reason we sometimes identify the units of the neural maximal clique solver with the vertices of the underlying graph.

Given an appropriate parameter settings the neural maximal clique solver operates as follows: An initial activation is imposed on the network. Finding a maximum clique then proceeds in accordance with eqn. (1) until the system reaches a stable state. The stable states correspond to the maximal cliques of G. In the ideal case a maximum clique is found. The clique size can be read out by counting the number of units with activation $x_i(t) > 0$.

4 Parameter Settings

In this section we derive bounds on the excitatory/inhibitory weights of the WTA network and on the initial pseudo-temperature of the transfer function.

[1] More precisely from the graph-theoretic point of view: uniquely up to isomorphism.

At the end of this section we present a termination criterion which recognizes a feasible stable equilibrium state.

To simplify the following considerations we introduce some useful technical terms: Let $G = (V, E)$ be a graph. Then according to the last section G uniquely determines the topology of a neural maximal clique solver. We call the number $\deg_E(i)$ of excitatory connections of unit i the *excitatory degree of* i. With $\deg_E = \max\{\deg_E(i) \mid 1 \leq i \leq n\}$ we denote the *excitatory degree* of the maximal clique solver. Similarly, $\deg_I(i) := n - \deg_E(i)$ is the *inhibitory degree* of i and $\deg_I = \max\{n - \deg_E(i) \mid 1 \leq i \leq n\}$ is the *inhibitory degree of the network*. With E_i (I_i) we denote the set of all units j which are excitatory (inhibitory) connected to unit i.

A Lower Bound for w_I

Given appropriate parameter settings the network performs a gradient descent with respect to some energy function E where the minima of E correspond to maximal cliques of the underlying graph. The inhibitory connections with weight w_I constitute the constraints of the problem and therefore are responsible to ensure feasibility of the solution. Proposition 1 gives a lower bound for w_I to ensure that the minima of the energy function E correspond to maximal cliques.

Proposition 1. *Let* $\boldsymbol{h}^\tau(t) \in \{0,1\}^n$ *be a stable equilibrium point. If* $w_I > \deg_E \cdot w_E$ *then* $\boldsymbol{h}^\tau(t)$ *is a feasible solution.*

Proof. By assumption the vector $\boldsymbol{h}^\tau(t)$ is a vertex point of the unit hypercube, since $\boldsymbol{h}^\tau(t)$ is a stable point. Now assume that $\boldsymbol{h}^\tau(t)$ is a infeasible solution. Then there exist units r and s with $w_{rs} = w_{sr} = -w_I$ and $h_r^\tau(t) = h_s^\tau(t) = 1$. We find

$$\begin{aligned} x_r(t+1) &= x_r(t) + w_E \sum_{j \in E_r} h_j^\tau(t) - w_I \sum_{k \in I_r} h_k^\tau(t) \\ &\leq x_r(t) + \deg_E \cdot w_E - w_I \cdot h_s^\tau(t) \\ &= x_r(t) + \deg_E \cdot w_E - w_I \\ &< x_r(t) \end{aligned}$$

Thus $x_r(t+1) < x_r(t)$ which is a contradiction to the assumption that $\boldsymbol{h}^\tau$ is a stable point. □

An Upper Bound for w_E

Prop. 1 ensures that a stable vertex point is a local minimum of the energy function E and corresponds to a maximal clique. Next we give an upper bound of w_E to ensure that the network performs a gradient descent towards a minimum.

Proposition 2. *Let* $E(t) = -\frac{1}{2} \sum_i \sum_{j \neq i} w_{ij} h_i^\tau(t) h_j^\tau(t)$ *and* $w_I > \deg_E \cdot w_E$. *If*

$$w_E < \frac{2 \cdot \tau}{\deg_I(n - \deg_E)} \tag{3}$$

then update rule (1) minimizes E.

Proof. Let $\Delta_E(t) := E(t+1) - E(t)$, $\boldsymbol{u} = \boldsymbol{h}^{\boldsymbol{\tau}}(t+1)$, $\boldsymbol{u'} = \boldsymbol{h}^{\boldsymbol{\tau}}(t)$, $\boldsymbol{\Delta} := \boldsymbol{u} - \boldsymbol{u'}$, and $\boldsymbol{I}$ be the identity matrix. Then

$$\begin{aligned}\Delta_E(t) &= -\frac{1}{2}\big(\langle \boldsymbol{u}, \boldsymbol{W}\boldsymbol{u}\rangle - \langle \boldsymbol{u'}, \boldsymbol{W}\boldsymbol{u'}\rangle\big) \\ &= -\frac{1}{2}\big(\langle \boldsymbol{\Delta}, \boldsymbol{W}\boldsymbol{\Delta}\rangle - 2\langle \boldsymbol{\Delta}, \boldsymbol{W}\boldsymbol{u'}\rangle\big) \\ &\leq -\frac{1}{2}\big(\langle \boldsymbol{\Delta}, \boldsymbol{W}\boldsymbol{\Delta}\rangle - 2\tau\langle \boldsymbol{\Delta}, \boldsymbol{\Delta}\rangle\big) \\ &= -\frac{1}{2}\langle \boldsymbol{\Delta}, (\boldsymbol{W} + 2\tau\boldsymbol{I})\boldsymbol{\Delta}\rangle\end{aligned}$$

where the inequality in the third line follows from Lemma 1. Thus $\Delta_E(t) \leq 0$ if $\boldsymbol{W} + 2\tau\boldsymbol{I}$ is positive definite. This condition is satisfied if

$$2\tau > \sum_{j \not= i} |w_{ij}| \tag{4}$$

for all i. Note, that 2τ are the diagonal elements of $\boldsymbol{W} + 2\tau\boldsymbol{I}$ since $w_{ii} = 0$ for all i. Let $\delta_i := \deg_I(i)$ be the inhibitory degree of unit i. Then

$$\sum_{j \not= i} |w_{ij}| = (n - \delta_i)w_E + \delta_i w_I \tag{5}$$

for all i. Since $w_I > w_E$ by assumption the sum becomes maximal if $\delta_i = \deg_I$. Thus we have

$$\begin{aligned}2\tau &> (n - \deg_I)w_E + \deg_I \cdot w_I \\ &> (n - \deg_I)w_E + \deg_I \cdot \deg_E \cdot w_E \\ &= (n - \deg_E)\deg_I \cdot w_E\end{aligned}$$

Solving the last equation for w_E yields the assumption. □

To complete the proof of Prop. 2 it is left to show Lemma 1.

Lemma 1. *Let* $\boldsymbol{\Delta} := \boldsymbol{h}^{\boldsymbol{\tau}}(t+1) - \boldsymbol{h}^{\boldsymbol{\tau}}(t)$. *Then*

$$\langle \boldsymbol{\Delta}, \boldsymbol{W}\boldsymbol{h}^{\tau}(t)\rangle \geq \tau\langle \boldsymbol{\Delta}, \boldsymbol{\Delta}\rangle \tag{6}$$

Proof. Let $\Delta_i := h_i^{\tau}(t+1) - h_i^{\tau}(t)$. We consider three cases:

- *Case 1* $\Delta_i > 0$: From $\Delta_i = h_i^{\tau}(t+1) - h_i^{\tau}(t) > 0$ follows $h_i^{\tau}(t+1) > 0$ and $h_i^{\tau}(t) < 1$. We have $x_i(t+1) \geq \tau \cdot h_i^{\tau}(t+1)$ and $x_i(t) \leq \tau \cdot h_i^{\tau}(t)$. Using both inequalities and update-rule (1) we obtain

$$\begin{aligned}\tau\Delta_i &= \tau \cdot h_i^{\tau}(t+1) - \tau \cdot h_i^{\tau}(t) \leq x_i(t+1) - \tau \cdot h_i^{\tau}(t) \\ &= x_i(t) + \sum_{j \not= i} w_{ij} h_j^{\tau}(t) - \tau \cdot h_i^{\tau}(t) \\ &\leq \sum_{j \not= i} w_{ij} h_j^{\tau}(t)\end{aligned}$$

  Since $\Delta_i > 0$ we have $\Delta_i \cdot \sum_{j \not= i} w_{ij} h_j^{\tau}(t) \geq \tau \cdot \Delta_i^2$.

- *Case 2* $\Delta_i = 0$: In this case $\Delta_i \cdot \sum_{j \not\leftarrow} w_{ij} h_j^\tau(t) = 0 = \tau \cdot \Delta_i^2$ holds.
- *Case 3* $\Delta_i < 0$: With a similar argumentation like in case 1 we conclude

$$\tau \Delta_i \geq \sum_{j \not\leftarrow} w_{ij} h_j^\tau(t)$$

using the inequalities $x_i(t+1) \leq \tau \cdot h_i^\tau(t+1)$, $x_i(t) \geq \tau \cdot h_i^\tau(t)$ and update-rule (1). Clearly, $\Delta_i \cdot \sum_{j \not\leftarrow} w_{ij} h_j^\tau(t) \geq \tau \cdot \Delta_i^2$ holds.

In any case we find that $\Delta_i \cdot \sum_{j \not\leftarrow} w_{ij} h_j^\tau(t) \geq \tau \cdot \Delta_i^2$. Thus the assumption holds. □

A Lower Bound for τ_0

In general $\boldsymbol{h}^\tau = \mathbf{0}$ is a feasible but useless solution. If for all units i the inhibitory degree $\deg_I(i)$ is larger than $\deg_E(i)/\deg_E$ and if τ is too small than the network converges to $\mathbf{0}$ within one time step. The following Prop. 3 gives a lower bound for the initial pseudo-temperature τ_0 to avoid a $\mathbf{0}$-solution.

Proposition 3. *Let $\tau_0 > 0$ be the initial pseudo-temperature. Suppose that the initial activation $x_i(0) = x_0$ with $0 < x_0 < 1$ for all units $1 \leq i \leq n$. If*

$$\tau_0 > \deg_I \cdot w_I - (n - \deg_I) \cdot w_E \tag{7}$$

then $x_i(1) > 0$ for all units i.

Proof. To avoid that the activation of all units are set to 0 at $t = 1$ we require according to the update-rule (1)

$$x_i(1) = x_0 + (n - \delta_i) \cdot w_E \cdot h_i^{\tau_0}(0) - \delta_i \cdot w_I \cdot h_i^{\tau_0}(0) > 0$$

for all units i where $\delta_i = deg_I(i)$ is the inhibitory degree of unit i. Since $1 > x_0 > 0$ and by definition of $h_i^{\tau_0}(0)$ we find

$$\begin{aligned} & x_0 + (n - \delta_i) \cdot w_E \cdot \frac{x_0}{\tau_0} - \delta_i \cdot w_I \frac{x_0}{\tau_0} > 0 \\ \Leftrightarrow\ & \tau_0 > \delta_i \cdot w_I - (n - \delta_i) \cdot w_E \end{aligned}$$

for all units i. The assumption follows from $\delta_i \cdot w_I - (n - \delta_i) \cdot w_E \leq \deg_I \cdot w_I - (n - \deg_I) \cdot w_E$. □

Termination Criterion

The network is in a stable state when the output $\boldsymbol{h}^\tau(t)$ does not change for all $t \geq t_0$ for some t_0. The following Lemma 2 gives a termination criterion which satisfies the output stability of the network.

Lemma 2. *The dynamical system given by (1) is stable, if for all units i the following conditions are satisfied:*

1. $h_i^\tau(t) \in \{0, 1\}$
2. $h_i^\tau(t+1) = 1 \rightarrow x_i(t+1) \geq x_i(t)$
3. $h_i^\tau(t+1) = 0 \rightarrow x_i(t+1) \leq x_i(t)$

Proof. Simply follows by induction using the inequality $w_I > deg_E \cdot w_E$. □

5 Experiments

In order to assess the performance of the proposed WTA heuristics, extensive simulations were carried out. We employed Steepest Descent (SD) as described in [10] for comparison. We found Steepest Descent attractive because it is extremely fast as a greedy emulation formulated in terms of an energy minimization procedure.

Test Graphs

We have tested the algorithms on the following types of graphs which are also commonly used for benchmark test within the neural network literature [4,7,10].

1. *Random Graphs*: Let V be a vertex set consisting of n elements and $p \in [0,1]$ be a fixed number called *success probability* or *edge probability*. A p-random n-vertex graph is intuitively generated as follows: For each pair $\{i,j\} \in [V]^2$ we decide by some random experiment whether or not $\{i,j\}$ shall be an edge in G. These experiments are performed independently with fixed probability of success p, i.e. the probability of accepting $\{i,j\}$ as an edge for G is equal to p.
   Generating random graphs with varying edge probability p for fixed number of vertices n is a simple and powerful method to generate graphs with various distinct graph properties like (non)-connectivity, (non)-planarity, (non)-existence of Hamiltonians, etc. However, the clique size of *almost all* random graphs is relatively small ($O(\log(n))$).
2. *Random Graphs with large Cliques*: These graphs are p-random n-vertex graphs where additionally a large clique is inserted. Let n be the number of vertices of a graph G and $\alpha \in [0,1]$ a fixed number. Then $[n^\alpha]$ vertices of G are randomly selected and interconnected to a clique of size $[n^\alpha]$ where $[.]$ denotes the round-function.
   Theses class of graphs are of interest, because they also have various distinct graph properties like p-random n-vertex graphs and additionally circumvent the restriction of graphs with small clique numbers.
3. *k-Random Cliques Graphs*: A k-random cliques graph G is constructed by randomly generating k cliques of various size and taking their union: Let V be the vertex set of a graph G with n elements. For each clique C_i $(1 \leq i \leq k)$ and for each vertex $v \in V$ $(1 \leq v \leq n)$ we decide by a Bernoulli trial with probability of success $p = 0.5$ whether or not vertex v is a member of clique $C_i = (V_i, E_i)$. By taking the union of all cliques $C_1, \ldots, C_k$ we obtain a k-random cliques graph $G = (V, E)$ with $V_i \subseteq V$ and $E = E_1 \cup \cdots \cup E_k$.
   Due to the construction process just described, k-random cliques graph have a wider range of different clique sizes than random graphs. This suggests that the maximum clique problem for such graphs is *harder*.

Experiment Set-Up and Control

The sizes of random graphs and k-random graphs considered in our experiments were $n = 100, 250, 500, 750, 1000$. The chosen edge probabilities for the first

random graphs series were $p = 0.1, 0.25, 0.5, 0.75, 0.9$ and for the second random graphs series with large cliques we concentrated on $p = 0.5$, $\alpha = 0.5, 0.75, 0.9$. For the k-random cliques series we selected $k = n/25, n/10, n/5$. For each pair (n, p), (n, α) and (n, k) 100 different graphs with the specified parameters were generated. Overall we employed 6500 graphs for testing the WTA algorithm on the maximum clique problem.

Settings of the WTA Algorithm

At $t = 0$ we initialize the activation $x_i(0)$ of each unit i with 0.1. During evolution we perturb instable equilibria with small random noise. To speed-up convergence of the algorithm and in order to push the output vector $\boldsymbol{h}^{\boldsymbol{\tau}}$ to a vertex of the hypercube we apply the following geometric annealing schedule: At $t = 0$ we initialize the pseudo-temperature τ with $\tau_0 + \varepsilon$ where $\varepsilon > 0$ is a small constant to fulfill inequality in accordance with Prop. 3. At each iteration τ is decreased by $\tau_{t+1} = a_t \cdot \tau_t$ where $a_t = 0.9$ if $t = 1$ and $a_t = 0.2$ if $t > 1$.

Results and Discussion

Table 1 summarize the results of our test series for p-random n-vertex graphs, Table 2 for random graph with large cliques, and Table 3 for k-random cliques graphs.

Each row of Table 1, 2, and 3 shows the size $|G|$ of 100 test graphs in the first column. The second column is reserved for the specific parameters of the corresponding classes of test graphs considered in our empirical study. In particular, the second column of Table 1 shows the density p for p-random n-vertex graphs, the second column of Table 2 shows the parameter α for random graphs with large cliques, and finally the second column of Table 3 shows the number k for k-random cliques graphs. The last four columns present the average clique size and the average number of iterations of SD and the WTA algorithm.

For p-random n-vertex graphs the average clique sizes found in different runs by the WTA algorithm are in most cases only slightly larger than the ones found by SD. An exception are highly dense graphs ($p = 0.9$) where the WTA algorithm outperforms SD with respect to average clique size and average number of iterations. For both algorithm the average number of iterations increases with the number of vertices and with the edge probability. For fixed number of vertices $|G|$, SD is always significantly faster than the WTA algorithm for sparse random graphs. But with increasing edge probability p the average computation time[2] of SD increases faster than the average computation time of the WTA algorithm until SD is much slower than WTA for dense random graphs. If we consider the average clique size found and the average computation time both algorithms perform almost equally. Since it is known that greedy heuristics like SD work well on random graphs [7] we can conclude that the same holds for the WTA algorithm.

[2] The computation time is measured by the number of iterations.

Table 1. Results of test series on p-random n-vertex graphs

Graphs		SD	WTA	SD	WTA
$\|G\|$	p	clique size		iterations	
100	0.10	2.6	2.7	3.6	11.5
100	0.25	3.8	3.9	4.8	11.4
100	0.50	6.3	7.5	7.3	13.7
100	0.75	12.2	14.1	13.2	14.2
100	0.90	23.5	27.8	24.5	15.3
250	0.10	3.0	3.0	4.0	11.9
250	0.25	4.5	4.5	5.5	13.6
250	0.50	7.7	7.7	8.7	16.0
250	0.75	15.2	15.5	16.2	22.6
250	0.90	31.5	36.0	32.5	27.9
500	0.10	3.3	3.4	4.3	12.7
500	0.25	5.0	5.0	6.1	13.4
500	0.50	8.8	8.9	9.8	18.4
500	0.75	17.6	19.5	18.6	23.5
500	0.90	38.2	43.5	39.2	26.9
750	0.10	3.5	3.5	4.5	13.9
750	0.25	5.2	5.3	6.2	14.9
750	0.50	9.3	9.3	10.3	18.4
750	0.75	18.8	18.8	19.8	27.8
750	0.90	41.7	45.5	42.7	38.9
1000	0.10	3.6	3.7	4.6	13.9
1000	0.25	5.5	5.4	6.5	15.2
1000	0.50	9.5	9.6	10.5	19.2
1000	0.75	20.0	20.1	21.0	28.9
1000	0.90	44.2	46.9	45.2	42.0

On random graphs with large cliques with $\alpha = 0.75$, and $\alpha = 0.9$ the WTA algorithm significantly outperforms SD with respect to solution quality and computation time. Only for $\alpha = 0.5$ the overall performance of SD is slightly better than the performance of WTA. In this case the average clique size is similar but the computation time of SD is much shorter.

In our last test series using k-random cliques graphs the SD heuristics breaks down completely and yields noticeable worse solutions than the WTA algorithm. In contrast to the test series on random graphs SD requires much more time to find a solution. Starting from a relatively high level the computation time of SD increases linearly with the size of the test graphs for the following reason: For each of the k cliques the expected clique size is $n/2$. Thus if SD finds one of the k cliques its expected number of iterations will be $n/2$.

The results of all three test series suggest that in the average the WTA algorithm is superior to SD in terms of quality of solution and computational effort. Note, that in all 6500 instances, the WTA algorithm computed a non-trivial feasible solution.

Table 2. Results of test series on random graphs with large cliques

Graphs		SD	WTA	SD	WTA
$\|G\|$	α	clique size		iterations	
100	0.50	7.4	8.0	8.4	13.7
100	0.75	16.7	31.0	17.7	9.0
100	0.90	45.7	63.0	46.7	8.6
250	0.50	10.3	10.2	11.3	18.6
250	0.75	29.3	34.5	30.3	25.4
250	0.90	112.3	143.0	113.3	9.5
500	0.50	13.0	12.1	14.0	21.3
500	0.75	46.1	105.0	47.1	11.1
500	0.90	200.1	268.0	201.1	10.0
750	0.50	14.3	14.3	15.3	23.6
750	0.75	63.5	63.8	64.5	60.1
750	0.90	229.6	386.0	230.6	10.0
1000	0.50	17.4	16.8	18.4	25.9
1000	0.75	69.3	75.4	76.4	75.7
1000	0.90	305.7	501.0	306.7	11.0

Table 3. Results of test series on k-random cliques graphs

Graphs		SD	WTA	SD	WTA
$\|G\|$	k	clique size		iterations	
100	4	24.1	38.7	25.1	10.5
100	10	32.8	39.6	33.8	11.0
100	20	42.6	49.9	43.6	12.6
250	10	84.6	92.8	85.6	13.7
250	25	102.2	119.5	103.2	16.8
250	50	179.7	190.2	180.7	22.3
500	20	165.4	195.5	166.4	18.6
500	50	326.5	355.6	327.5	28.7
500	100	485.4	487.3	486.4	20.1
750	30	310.1	359.0	311.1	21.2
750	75	637.7	655.6	638.7	37.8
750	150	749.4	749.4	750.4	7.5
1000	40	473.5	544.2	474.5	23.7
1000	100	960.9	966.2	961.9	30.1
1000	200	999.9	999.9	1000.9	5.4

6 Conclusion

We proposed a high speed algorithm by means of a simple neural network. In a mathematical analysis we derived bounds for the parameter settings to ensure that the network performs an energy descent to a maximal clique[3]. Thus only the optimal parameter settings of the cooling schedule is left as an open problem.

Since we focused on rapid solutions for large scaled problems, we employed a fast annealing schedule on the WTA algorithm. In the average the proposed WTA algorithm provides suboptimal approximations of the classical maximum clique problem within short time. Since neural networks are a massively parallel computational model the WTA algorithm is especially suited for parallel implementation. Due to its high speed and the theoretical justification of its parameter settings the proposed algorithm may be an attractive alternative to more accurate but noticeably slower heuristics like Mean-Field Annealing or simulated annealing for large-scaled applications. In an extensive empirical study over 6500 randomly generated graphs with up to 1000 vertices we have demonstrated the effectiveness of the proposed WTA algorithm with respect to quality of approximations and computational effort.

[3] Note, that a clique is called maximal if it is not contained in any larger clique. The vertex set of a maximal clique does not necessarily have maximum cardinality. In general a maximal clique is not a maximum clique but a maximum clique is always maximal. The WTA algorithm guarantees solutions which correspond to maximal cliques.

References

1. E. Balas and C.S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal of Computing*, 15(4):1054–1068, 1986.
2. I.M. Bomze, M. Budinich, P.M. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P.M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 4, pages 1–74. Kluwer Academic Publishers, Boston, MA, 1999.
3. U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Approximating clique is almost np-complete. In *Proc. 32nd Ann. IEEE Symp. Found. Comput. Sci.*, pages 2–12, 1991.
4. N. Funabiki and S. Nishikawa. Comparisons of energy-descent optimization algorithms for maximum clique. *IEICE Trans. Fundamentals*, E79-A(4):452–460, 1996.
5. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.
6. L. Gerhards and W. Lindenberg. Clique detection for nondirected graphs: two new algorithms. *Computing*, 21:295–322, 1979.
7. S. Homer and M. Peinado. On the performance of polynomial-time clique approximation algorithms. In D.S. Johnson and M. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1996.
8. J.J. Hopfield and D.W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.
9. R. Horaud and T. Skordas. Stereo correspondence through feature grouping and maximal cliques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1168–1180, 1989.
10. A. Jagota. Approximating maximum clique with a Hopfield network. *IEEE Trans. Neural Networks*, 6:724–735, 1995.
11. B.J. Jain and F. Wysotzki. Distance-based classification of structures within a connectionist framework. In R. Klinkenberg et al., editor, *Proceedings Fachgruppentreffen Maschinelles Lernen*, 2001.
12. J.E. Lecky, O.J. Murphy, and R.G. Absher. Graph theoretic algorithms for the pla folding problem. *IEEE Transactions on Computer Aided Design*, 8(9):1014–1021, 1989.
13. H. Ogawa. Labeled point pattern matching by delauney triangulation and maximal cliques. *Pattern Recognition*, 19(1):35–40, 1986.
14. M. Pelillo, K. Siddiqi, and S.W. Zucker. Matching hierarchical structures using association graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11):1105–1120, 1999.
15. K. Schädler and F. Wysotzki. Application of a neural net in classification and knowledge discovery. In M. Verleysen, editor, *Proc. ESANN'98*, pages 117–122. D-Facto, Brussels, 1998.
16. K. Schädler and F. Wysotzki. Comparing structures using a Hopfield-style neural network. *Applied Intelligence*, 11:15–30, 1999.
17. K.A. Smith. Neural networks for combinatorial optimisation: A review of more than a decade of research. *INFORMS Journal on Computing*, 11(1):15–34, 1999.

Augmenting Supervised Neural Classifier Training Using a Corpus of Unlabeled Data

Andrew Skabar

School of Information Technology
International University in Germany
D-76646 Bruchsal Germany
andrew.skabar@i-u.de

Abstract. In recent years, there has been growing interest in applying techniques that incorporate knowledge from unlabeled data into systems performing supervised learning. However, disparate results have been presented in the literature, and there is no general consensus that the use of unlabeled examples should always improve classifier performance. This paper proposes a method for incorporating a corpus of unlabeled examples into the supervised training of a neural network classifier and presents results from applying the technique to several datasets from the UCI repository. While the results do not provide support for the claim that unlabeled data can improve overall classification accuracy, a bias-variance decomposition shows that classifiers trained with unlabeled data display lower bias and higher variance than classifiers trained using labeled data alone.

1 Introduction

In recent years, there has been growing interest in applying techniques that incorporate knowledge from unlabeled data into systems performing supervised learning. The motivation for this is that in many learning scenarios a small amount of labeled data is accompanied by a large corpus of unlabeled data. Intuitively, by utilizing the additional context provided by the unlabeled data, it may be possible to achieve higher generalization accuracy than can be achieved by using supervised learning on the labeled training data alone.

Various techniques have been proposed for incorporating unlabeled data into supervised learning tasks. One family of such techniques is based on the Expectation-Maximization (EM) algorithm [1,2,3]. A classifier is first trained using the available labeled examples, and probabilistically assigns *soft* labels to the unlabeled data. It then trains a new classifier using the labels for all examples, and iterates until convergence. While EM is a hill climbing method, and is thus guaranteed to find only a local maximum, the major problem is that this local maximum is often poor. This is because in the first assignment of soft labels, a large number of the unlabeled examples will be assigned to classes quite confidently, based on a possibly poor initial model fitted to the

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 174-185, 2002.

labeled examples. These artificially labeled examples can outweigh the labeled training examples, leading to a model that might exhibit worse predictive performance than the initial one, and converge quickly to a poor local maximum.

A second approach is that of *co-training* [4][5]. Co-training methods are based on the intuition that different supervised learning algorithms may complement each other because they use different representations for their hypotheses, and use the provided labeled data in different ways. Co-training works by using one learner to label examples which are then fed to the other learner, and vice-versa. The main problem is that in practice, each of the individual learners will contribute some degree of noise, and this will cause co-training to follow a sub-optimal path.

Two other methods for incorporating unlabeled examples into supervised learning are *transduction* [6] and *maximum entropy discrimination* [7]. Each of these attempts to maximize the classification margin on both labeled and unlabeled data while classifying the labeled data as accurately as possible. While transduction sets the margin directly, maximum entropy discrimination optimises a distribution of margins to lie as close as possible to a prior.

Despite the flurry of interest in this area, there is some degree of uncertainty in the literature regarding the utility of incorporating unlabeled data into supervised classification tasks. Most of the papers which claim an improvement in performance resulting from the use of unlabeled data also note cases in which degradation in performance was observed. For example, Shahshahani and Landgrebe (1994) [8] used the EM algorithm to examine the effect that unlabeled examples can have on reducing the effects of the Hughes phenomenon[1]. However, while the performance of their classifier increased with the use of unlabeled data, they note that performance decreased when the number of features was small. Similarly, Baluja (1998) [9] and Nigam *et al* (2000) [3] (both of which used EM with Naive Bayes classifiers) each claim that there were situations in which their respective classifiers displayed improved performance when trained in the presence of unlabeled data. However, in each of these studies degradation in performance was observed when the number of labeled examples was relatively large. Cozman and Cohen (2001) [10] also report on cases in which unlabeled data caused degradation in performance. Thus, there appears to be little concrete evidence for or against the claim that unlabeled data can, in a general sense, assist in reducing the overall error on supervised classification tasks. Also, most papers apply the respective techniques to novel datasets, and there are no results reported in the literature on the application of such techniques to standard datasets such as those from the UCI repository. This adds to the difficulty of gauging whether there is any real expected advantage in incorporating unlabeled examples into supervised classification tasks.

This paper proposes an alternative method by which unlabeled data may be incorporated into supervised learning tasks. The method is based on using a feedforward neural network model to estimate a posterior probability function for each class in the classification domain, and assigning a test example to the class for which the poste-

[1] Increasing the number of attributes in a classification task requires that the number of parameters in the model be increased. This leads to an increase in variance, and eventually a degradation in classification performance. This is known as the *Hughes phenomenon*.

rior probability of membership is highest. The role of unlabeled data can be best understood by comparing the approach with methods based on non-parametric density estimation. In these methods, class conditional and class unconditional probability density functions (pdfs) are estimated from the training data, and these pdfs are combined using Bayes' theorem and a set of priors to obtain a posterior probability function. Unlabeled examples can be used to obtain a more precise estimation of the unconditional probability density than would be possible using only labeled examples, and intuitively this would be expected to lead to some improvement in classification performance. However, rather than calculating class conditional and class *un*conditional pdfs and combining these with a prior using Bayes' theorem, a neural network is here used to represent the posterior probability functions directly. Nevertheless, the role of unlabeled examples is similar—they provide additional context that may lead to an improvement in classifier performance.

The remainder of the paper is organized as follows. Section 2 shows how neural networks can be used to represent posterior probabilities. Section 3 describes how a network can be trained to represent a posterior probability function for some target class from a training set consisting of labeled examples from that target class together with a corpus of unlabeled examples. This is then generalized to multi-class classification. Section 4 describes the experimental design and presents results and analysis of applying the method to several datasets from the UCI repository. Section 6 concludes the paper.

2 Representing Posterior Probabilities Using Neural Networks

While neural networks are often applied to function approximation tasks in which the dependent variable is continuous-valued, they can also be applied when the dependent variable is binary. In the latter case it can be shown that under an appropriate choice of error reduction function, the network outputs can be interpreted as posterior probabilities [11,12].

Assume a set D of binary-valued examples whose attribute values are described by a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$. Suppose that the training examples can be represented by some unknown binary function $f(\boldsymbol{x})$ that maps $\boldsymbol{x}$ onto its target value d, where d is 0 or 1. The objective is to learn a target function h: $X \rightarrow [0,1]$ such that $h(\boldsymbol{x}) = P(f(\boldsymbol{x}) = 1)$. Thus, $h(\boldsymbol{x})$ is a probabilistic function whose output is the *probability* that $f(\boldsymbol{x}) = 1$. The function $h(\boldsymbol{x})$ can be modelled using a feedforward neural network with a single output neuron. Because the network output is to represent a probability, the output values should be bounded between 0 and 1, and this can be achieved using a sigmoidal activation function.

The error function to be minimized can be derived using maximum likelihood considerations. Suppose that an example with input vector $\boldsymbol{x}_i$ is drawn randomly from the training set. By definition of $h(\boldsymbol{x})$, the probability that $\boldsymbol{x}_i$ has a target output of 1 is $h(\boldsymbol{x}_i)$, and the probability that is has a target output of 0 is $1 - h(\boldsymbol{x}_i)$. The probability of observing the correct target value, given hypothesis h, can therefore be expressed as

$$P(d_i \mid \mathbf{x}_i) \propto h(\mathbf{x}_i)^{d_i} (1 - h(\mathbf{x}_i))^{1-d_i} \tag{1}$$

where d_i is the target output corresponding to input vector $\boldsymbol{x}_i$. The maximum likelihood hypothesis, h_{ML}, is defined as the hypothesis h that results in the highest probability of observing the given data D. Assuming that the examples in the training set are independent, the maximum likelihood function can therefore be expressed as

$$h_{ML} \equiv \arg\max_{h \in H} \prod_{i=1}^{m} h(\mathbf{x}_i)^{d_i} (1 - h(\mathbf{x}_i))^{1-d_i} \tag{2}$$

where m is the number of examples. Using the fact that $\ln p$ is a monotonic function, h_{ML} can be expressed as a minimization:

$$h_{ML} \equiv \arg\min_{h \in H} \sum_{i=1}^{m} \{-d_i \ln h(\mathbf{x}_i) - (1 - d_i) \ln(1 - h(\mathbf{x}_i))\} \tag{3}$$

The term appearing in braces in Equation 3 is commonly referred to as 'cross-entropy', and it can be shown that the back-propagation error term for cross-entropy error reduction with a sigmoidal output activation function is $\delta = (d_i - h(\mathbf{x}_i))$.

3 Classifier Learning Assisted with Unlabeled Data

This section first describes how a neural classifier can be trained to represent a posterior probability function for a target class given some positively labeled examples of that target class together with a corpus of unlabeled data. This is then generalized to simultaneously learning probability functions for each class in the classification domain.

3.1 Single-Class Classifier Learning

Single-class classifier learning is the problem of learning a classifier from a set of training examples in which only examples of the target class (i.e., *positive* examples) are present. From an inductive learning perspective, this is a tricky problem because in the absence of counter-examples of the target concept it is difficult to prevent unrestricted over-generalization from occurring. However, by using the context provided by unlabeled data, it can be shown that single-class learning becomes feasible.[2]

The most common approaches to single-class classifier learning tasks are based on probability density estimation [13][14]. Typically, a probability density function (pdf)

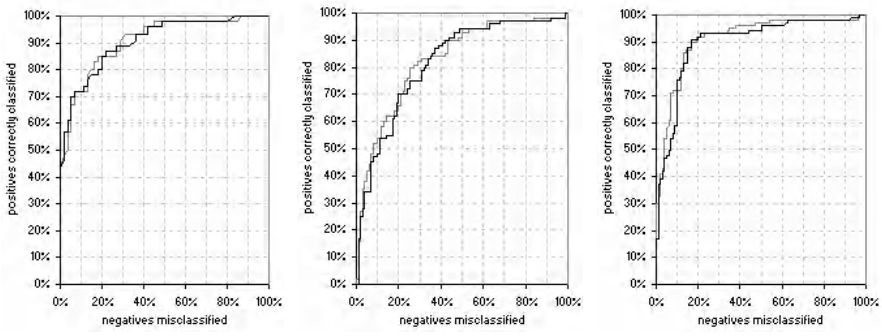
[2] The unlabeled examples will contain examples as well as *counter*-examples of the target concept. Thus, in this sense the learning is not strictly *single*-class. However, as the *labeled* training examples come from a single class, we continue to use this terminology.

is estimated from the training examples using an appropriate method (e.g., Parzen window [15]), and a probability threshold is selected. Input vectors which cause this threshold to be exceeded are classified as belonging to the target class, and others are rejected. While this implicitly assumes a uniform distribution of background examples, an alternative is to estimate the *un*conditional pdf in addition to the *class*-conditional pdf, and to combine these with priors using Bayes' theorem to arrive at an estimate of the posterior probability function. The unconditional pdf is based on all training examples, irrespective of their class membership. Unlabeled examples, if available, can also be used in the calculation of this density function.

However, there are several problems with approaches based on density estimation. Firstly, it is often very difficult to obtain a reliable estimate of the pdf, particularly when the number of input dimensions is high and/or there are few training examples available. Secondly, the techniques often rely on assumptions of conditional independence between input variables. Also, density estimation can be difficult to apply in domains containing a combination of continuous and categorical attributes. An alternative approach, which does not suffer from the above difficulties, is to model posterior probability function directly, and one means of doing this is to use a neural network model described in the previous section.

Implicit in Sect. 2 was the assumption that examples with training labels of 1 and 0 represent respectively known (i.e., labeled) positive examples and known negative examples (i.e., counter-examples) of the target class. Consider a modified learning situation in which a target label of 1 is again used to denote a known positive, but in which a label of 0 represents an *un*labeled positive *or* (unlabeled) negative. In this case the output of a neural network trained on such a set using cross-entropy error reduction will represent the posterior probability that an input vector presented to the network is one of the *labeled* positive examples. In other words, the sum of network outputs over all examples in the training set will equal the number of labeled positives. Because there are positive examples amongst the unlabeled examples, the probability represented by a network trained in such a way will be less than the true probability that an example belongs to the target class. However, under the assumption that the labeled positives are sufficiently representative of positive examples appearing in the unlabeled data, it would be expected that the probability function—when thresholded appropriately—will still be able to discriminate to a sufficient degree of accuracy between positive and negative examples.

In previous work [16] this technique has been applied to several real world datasets, and the resulting ROC charts are shown in Fig. 1. The grey curves represent the performance of standard back-propagation, i.e., training performed using labeled examples from each class. The dark curves show the performance of the method described above. In this case, the same training examples were used as for conventional back-propagation except that—with the exception of 30% of the positive examples—the class labels of all training examples were hidden from the learner. Note that in each case, the classification performance achieved using labeled positives together with unlabeled data does not differ significantly from that achieved using conventional supervised learning.



(a) (b) (c)

Fig. 1. ROC analysis of neural classifier learning from a training set consisting of labeled positive examples of the target concept together with a corpus of unlabeled positive and negative examples. The dark curve represents classification performance using only labeled positive examples; the lighter curve represents performance for standard back-propagation. (a) Cleveland Heart Disease (b) PIMA diabetes (c) Australian Credit Dataset

3.2 Multi-class Classifier Learning

If it is possible to achieve such classification performance using only labeled examples of the target class, together with a corpus of unlabeled examples, then the question arises as to whether classification performance on *multi*-class classification problems can be improved through the use of unlabeled data. The approach described above for single-class classifier learning can easily be extended to multi-class classifier learning.

As described in Sect. 2, when learning posterior probabilities, the appropriate error function is cross-entropy. However, in the multi-class case, an additional requirement that the sum over all network outputs be equal to 1 must normally be imposed (i.e., an input example must belong to one of the classes). One choice of activation function that achieves this is the *softmax* function [17]. However, use of the softmax function in this context presents difficulties because the sum of outputs in this case will be less than 1. This is because the outputs in this case represent the posterior probability conditioned on the *labeled* examples, together with a possibly large set of unlabeled examples. Thus, the appropriate error reduction is once again cross-entropy.

Rather than having a single target output value associated with it, each training example must now have an associated target output *vector*. Assume a 1-of-*n* output encoding scheme in which each example has a target output vector in which a '1' appears in the position corresponding to the class to which the example belongs. Thus, each labeled example will have target vector consisting of a single '1', with all other values in the vector being '0'. The target class of unlabeled examples is represented with a target vector consisting of all '0's. Training then progresses as usual. The next section presents results from applying the method to several real world datasets.

4 Empirical Results

This section presents results from a series of experiments designed to compare the classification performance of the technique described above with that of a classifier trained using conventional supervised learning. Section 4.1 describes the experimental design, and the following subsection presents results and analysis.

4.1 Experimental Design

While we are obviously interested in determining whether supervised learning assisted with unlabeled examples can lead to an improvement in classification accuracy over conventional supervised learning, we would also like to obtain any additional information that may assist in comparing the two techniques. A useful tool for analysing the performance of supervised learning algorithms is the *bias-variance decomposition* [17]; that is, the decomposition of the overall generalization error into a *bias* term and a *variance* term. The bias measures the extent to which the average (over all sets of labeled training examples) of the network function differs from the desired function. That is, it is the systematic error in attempting to approximate the target function with the network function. Conversely, the variance measures the extent to which the network function is sensitive to the particular choice of training set. The original bias-variance decomposition due to Geman, Bienenstock & Doursat (1992) was developed for numeric regression tasks (i.e. quadratic loss function), and is not directly applicable to classifier learning [18]. As a result, several alternative formulations of the bias-variance decomposition for classifier learning have emerged [19,10,21,22]. The results presented in this paper are based on Kohavi and Wolpert's (1996) definition [19].

In order to estimate the bias and variance of a classifier on a particular dataset, we need to know the actual function being learned, and this is generally unavailable. To deal with this problem Kohavi and Wolpert [19] suggest holding out some of the data. This holdout data is treated as representing the true target function that we are trying to learn. A validation set is also required in order to determine a point at which to stop neural network training. Each dataset to be tested is divided into a training, validation, and holdout sets as follows:

1. Divide the entire set of training examples into two sets, **E** and **H**. The examples in set **E** are used for learning, while the examples in set **H** are used to perform the bias-variance decomposition.
2. Generate N training sets from **E** using the following procedure:
   (a) Select a fixed proportion of examples from **E** to constitute a set of *validation* examples, $\mathbf{E}_{\mathrm{V}}$.
   (b) Select a fixed proportion of the remaining examples in **E** to constitute a set of *labeled* training examples, $\mathbf{E}_{\mathrm{LT}}$.
   (c) Use the remaining examples in **E** as unlabeled training examples, $\mathbf{E}_{\mathrm{UT}}$.
3. Run the learning algorithm over the each of the N training sets described in 2 using **H** to estimate the bias-variance decomposition.

The utility of unlabeled examples could depend on a number of factors such as the absolute or relative number of labeled and unlabeled examples. On means of investigating this relationship would be to keep the size of $\mathbf{E}_{LT}$ constant and vary the size of $\mathbf{E}_{UT}$ (i.e., fix the amount of labeled training data and increase the amount of unlabeled examples). The problem here is that the effect due to increasing unlabeled examples may also be dependent on the absolute amount of labeled examples present. Thus, in order to account for the fact that the effect of unlabeled examples may depend on both $\mathbf{E}_{LT}$ and $\mathbf{E}_{UT}$, the proportion of training examples appearing with labels (i.e., $|\mathbf{E}_{LT}| / (|\mathbf{E}| - |\mathbf{Ev}|)$) was varied. In order to obtain a reliable estimate of the bias and variance 500 trials were performed for each case.

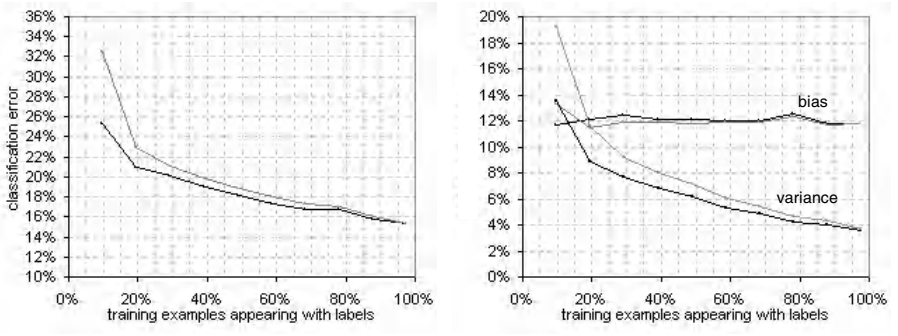
The use of a validation set in these experiments is crucial to obtaining a reliable comparison between the two methods. Any increase in classification accuracy that is observed using unlabeled data may be minimal, and for this reason it is imperative to ensure that the best possible performance is achieved when using labeled data alone. For example, if the stopping criterion was not based on a validation set, it could easily be the case that the degradation in performance resulting from overfitting exceeds any improvement in performance achieved using the unlabeled data.

4.2 Experimental Results and Analysis

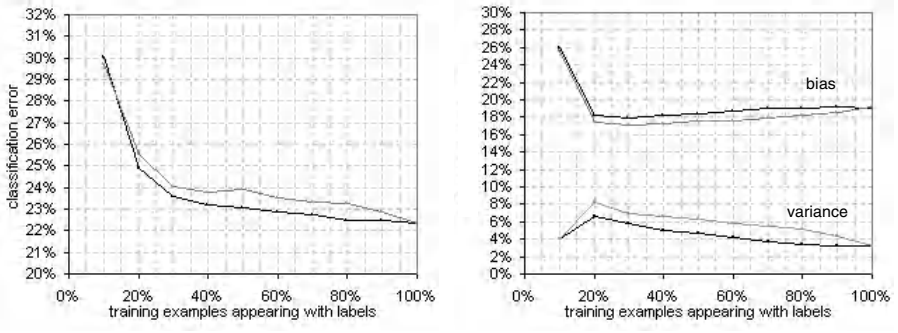
The results from the applying the above procedure to the same three datasets as reported in Fig. 1 are presented in Fig. 2. The charts on the left of the figure show the overall average classification error on the holdout data. The charts on the right show the decomposition of this error into a bias and variance term.

It is clear that the results presented in Fig. 2 do not provide any evidence that the use of unlabeled data can lead to an improvement in classifier performance. That is, the accuracy achieved by using the unlabeled data is consistently less than that achieved through the use of labeled data alone. While the overall classification error (with and without the use of unlabeled data) decreases as the proportion of training examples appearing with labels is increased, this is simply due to the fact that more labeled training examples are being used for training.

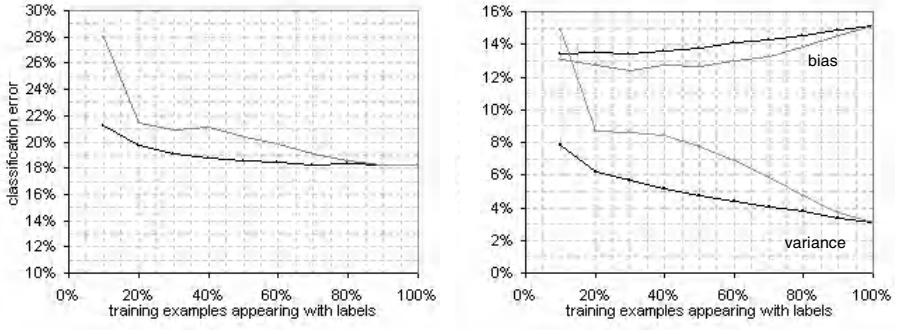
Although the use of unlabeled data does not lead to an improvement in classification performance, there appears to be a systematic difference between the two methods when the bias and variance characteristics are examined. In particular, the use of unlabeled examples appears to lead to a reduction in bias. This is especially apparent for the PIMA Diabetes and Australian Credit datasets, in which the bias obtained through the use of unlabeled examples is consistently less than that for conventional supervised learning. Unfortunately, the reduction in bias is accompanied by a significant increase in variance, and thus the overall generalization error is no lower than that for conventional learning. The observed fall in variance as the proportion of training examples appearing with labels is increased is to be expected since the (randomly selected) labeled training sets used in each of the N trials become more similar as the proportion of examples appearing with labels increases.



(a)



(b)



(c)

Fig. 2. Overall misclassification rates and bias-variance decomposition for three datasets from the UCI repository. Diagrams on the left represent overall error on holdout examples, and diagrams on the right show the decomposition into bias and variance. Solid lines represent learning using labeled examples only, and grey lines represent learning using unlabeled examples. (a) Cleveland Heart Disease dataset, (b) PIMA Diabetes dataset, (c) Australian Credit Dataset

It is often suggested that the main effect of using unlabeled data is to reduce variance. For example, according to Seeger, "Clearly one can come up with methods that reduce the *variance* of an estimator by employing unlabeled data, but there is no reason to believe that such modifications will work without introducing new bias" [23]. However, while it is certainly true that reducing variance will often be at the cost of increasing bias, it is not clear that the *main* effect of unlabeled data should be to reduce variance. The main argument for unlabeled data leading to a reduction in variance is that the unlabeled data places a further constraint on the learning, and that this additional constraint reduces the dependency on the particular set of (labeled) training examples. However, it could also be argued that the presence of unlabeled data increases the relative importance of the particular set of labeled examples used, and that this increased dependency on the labeled data makes the classifier more sensitive to those labeled examples, thus leading to an increase in variance. The results presented above are consistent with this line of reasoning, as in all three cases there was a clear increase in variance resulting from the use of unlabeled data. If the proposed technique is in fact able to significantly reduce the bias of a classifier, and if there are ways of reducing variance without affecting the bias, then it may be possible to combine such techniques to form a classifier that reduces both the bias and the variance.

As described in the introduction, there is some degree of uncertainty in the literature regarding the utility of incorporating unlabeled data into supervised classification tasks, and most reports of improvement in classification performance resulting from the use of unlabeled data also note cases in which a degradation in performance was observed. In reality, the question of whether it is generally possible to increase the performance of a supervised classifier using unlabeled examples can probably not be answered with a simple yes or no. The answer will depend on details of the classifier and the training method used, and also on the problem domain.

There is clearly a need for further work in this area. Areas to focus on should include: (i) identifying characteristics of datasets that can be used to determine *a priori* whether unlabeled examples are likely to have any utility in supervised classifier learning; (ii) comparing the utility of unlabeled data using various classification techniques on standard datasets; (iii) comparing the utility of unlabeled data using models of varying complexity, and, as already mentioned above, (iv) combining the proposed method with variance-reduction methods to construct a classifier that can consistently reduce both bias and variance.

5 Conclusions

A method has been proposed for incorporating a corpus of unlabeled examples into the supervised training of a neural network classifier. While results from applying the method to several difficult datasets from the UCI depository do not show a significant improvement in classifier accuracy, a bias/variance decomposition demonstrated that classifiers trained using the unlabeled examples displayed a lower bias, and higher variance, than classifiers trained using labeled data alone. An avenue worth

investigating is whether the proposed method can be combined with variance-reduction methods to construct a classifier that is able to simultaneously reduce both bias and variance.

References

1. Dempster, A.P., Laird, N.M. and Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society, Series B, 39, (1977), pp. 1-38.
2. Ghahramani, Z. & Jordan, I.: Supervised learning from incomplete data via an EM approach, in Advances in Neural Information Processing Systems 6. J.D. Cowan, G. Tesauro and J. Alspector (eds). Morgan Kaufmann Publishers, San Francisco, CA, (1994).
3. Nigam, K., McCallum, A.K., Thrun, S, & Mitchell, T.: Text classification from labeled and unlabeled documents using EM. Machine Learning, 39, (2000) pp. 103-134.
4. Blum, A. & Mitchell, T.: Combining labeled and unlabeled data with co-training, Proceeding of the Eleventh ANNUAL Conference on Computational Learning Theory (1998) pp. 92-100.
5. Goldman, S, and Zhou, Y.: Enhancing supervised learning with unlabeled data, Proceedings of International Conference on Machine Learning ICML 2000, (2000).
6. Vapnik, V.: Statistical Learning Theory. Wiley, (1998).
7. Jaakkola, T., Meila, M. & Jebara, T.: Maximum Entropy Discrimination, in nips, vol. 12, (1999), pp 470-476.
8. Shahshahani, B.M. and Landgrebe, D.A.: The effect of unlabeled samples in reducing the small size problem and mitigating the Hughes phenomenon, IEEE Transactions on Geoscience and Remote Sensing, 32(5), (1994) pp 1087-1095.
9. Baluja, S.: Probabilistic modeling for face orientation discrimination: Learning from labeled and unlabeled data, Neural and Information Processing Systems (NIPS) (1998).
10. Cozman, F.G and Cohen, I.: Unlabeled Data Can Degrade Classification Performance of Generative Classifiers, HP Labs Technical Report HPL-2001-234 (2001).
11. Richard, M.D. and Lippmann, R.P.: Neural network classifiers estimate Bayesian a posteriori probabilities, Neural Computation, 3(4) (1991) pp. 461-483.
12. White, H.: Learning in artificial neural networks: a statistical perspective. Neural Computation 1 (4), (1989), pp. 425-464.
13. Tarassenko, L., Hayton, P. & Brady, M.: Novelty detection for the identification of masses in mammograms, Proc. Fourth International IEEE Conference on Artificial Neural Networks, vol. 409, (1995) pp. 442-447.
14. Parra, L., Deco, G. & Miesbach, S.: Statistical independence and novelty detection with information preserving nonlinear maps, Neural Computation, vol. 8, (1996), pp. 260-269.
15. Duda, R.O. & Hart, P.E.: Pattern Recognition and Scene Analysis, John Wiley & Sons, New York, (1973).
16. Skabar, A.: Single-class classifier learning using neural networks: extracting context from unlabeled data, Artificial Intelligence and Applications (AIA2002), Malaga, Spain, 2002.
17. Bishop, C.: Neural Networks for Pattern Recognition, Oxford University Press, Oxford, (1995).
18. Geman, S., Bienenstock, E. & Doursat, R.: Neural Networks and the Bias/Variance Dilemma, Neural Computation, Vol. 4, (1992) pp. 1-58.

19. Kohavi, R. & Wolpert, D.H.: Bias plus variance decomposition for zero-one loss functions, Proceedings of the 13th International Conference on Machine Learning, Bari, Italy, (1996), pp. 275-283.
20. Breiman, L.: Bias, variance, and Arcing Classifiers. Technical Report 444486, Statistics Department, University of California, Berkeley, CA, (1996).
21. Kong, E.B. and Dietterich, T.G.: Error-correcting output coding corrects bias and variance, Proceedings of the 12th International Conference on Machine Learning, Tahoe City, CA, (1995) pp. 313-321.
22. Friedman, J.H.: On bias, variance, 0/1-loss, and the curse-of-dimensionality, Data Mining and Knowledge Discovery, Vol. 1, No. 1, Kluwer Academic Publishers. (1997) pp 55-77
23. Seeger, M.: Learning with labeled and unlabeled data. Technical Report, Institute of Adaptive and Neural Computation, University of Edinburgh, Edinburgh, UK, (2001).

Learning of Class Descriptions from Class Discriminations: A Hybrid Approach for Relational Objects

Peter Geibel[1], Kristina Schädler[2], and Fritz Wysotzki[1]

[1] Methods of Artificial Intelligence, Computer Science Department, Sekr. Fr 5–8
Technical University Berlin, Franklinstr. 28/29, D-10587 Berlin, Germany
{geibel,wysotzki}@cs.tu-berlin.de
[2] Pace Aerospace Engineering and Information Technology GmbH
Rotherstr. 20, D-10245 Berlin
kristina.schaedler@pace.de

Abstract. The paper addresses the question how learning class discrimination and learning characteristic class descriptions can be related in relational learning. We present the approach TRITOP/MATCHBOX combining the relational decision tree algorithm TRITOP with the connectionist approach MATCHBOX. TRITOP constructs efficiently a relational decision tree for the fast discrimination of classes of relational descriptions, while MATCHBOX is used for constructing class prototypes.
Although TRITOP's decision trees perform very well in the classification task, they are difficult to understand and to explain. In order to overcome this disadvantage of decision trees in general, in a second step the decision tree is supplemented by prototypes. Prototypes are generalised graphtheoretic descriptions of common substructures of those subclasses of the training set that are defined by the leaves of the decision tree. Such prototypes give a comprehensive and understandable description of the subclasses. In the prototype construction, the connectionist approach MATCHBOX is used to perform fast graph matching and graph generalisation, which are originally NP-complete tasks.

1 Introduction

Learning a classifier usually results in a hypothesis of one of the following two types. **Discriminating classifiers** decide whether the given object belongs to a certain class or not by using knowledge about class differences rather than thorough descriptions of each single class. Typical examples are decision trees that are in general fast to use and can be constructed efficiently. **Class descriptions** provide knowledge about the characteristic properties of the objects or typical instances of a class. Such descriptions are especially useful for knowledge discovery and user interaction.

Because characteristic descriptions tend to be structurally complex, it is often more efficient to learn class discriminations only. In the field of Inductive

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 186–201, 2002.

Logic Programming (ILP) class descriptions can be constructed using bottom up techniques like Plotkin's least general generalisation (*lgg*, s. [17]) for learning. An example is the system GOLEM [5] that uses a strong language restriction (deterministic clauses) for efficiency reasons. In this paper we will describe a different way of coping with complexity.

We combined TRITOP that constructs a relational decision tree for the fast discrimination of classes of relational descriptions, with the connectionist approach MATCHBOX that is used for constructing class prototypes[1]. Although TRITOP's decision trees perform very well in the classification task ([11]), they are difficult to understand and to explain. In order to overcome this disadvantage of decision trees in general, in a second step the decision tree is supplemented by prototypes that are generalised graph theoretic descriptions of common substructures of those subclasses of the training set that are defined by the leaves of the decision tree. Such prototypes give a comprehensive and understandable description of the subclasses. To our knowledge, this idea was first formulated for the system KATE ([16]), a frame oriented decision tree algorithm, though the author did not give an effective method for computing generalizations. We use the connectionist approach MATCHBOX to perform fast graph matching and graph generalisation in order to construct the prototypes.

The decision tree construction as well as the prototype construction in the domain of relational descriptions include originally NP-complete tasks. In this paper it is shown how this problem can be tackled in both steps. In TRITOP's decision tree algorithm, bottom up and top down induction techniques are combined for efficiently computing discriminating, i.e. small structural attributes (test graphs). The attribute construction is based on the concept of α-subsumption that will be shown to be especially useful for the combination with a prototype approach – in contrast to normal θ-subsumption [17].

Using decision trees for clustering is also proposed in [2,18]. Though the authors propose that a decision tree can already be seen as a (discriminating) description of clusters, their method is only applied to the propositional case, i.e. the relational structure of the examples is not taken into account at all. In contrast to the approaches KBG (relational clustering, s. [1]) and RIBL (Relational Instance Based Learning, [6]), learning and classification with TRITOP is not based on a similarity measure. In contrast to KBG, we use a more intelligible concept of generalization (RIBL does not compute generalizations).

This article is structured as follows. In the next section, an overview of TRITOP is given focusing on the features important for prototype construction. After that, the system MATCHBOX, a connectionist prototype builder, is introduced. Example prototypes for an artificial domain and for the mutagenicity dataset (e.g. [22,23]) show the ability of TRITOP/MATCHBOX to produce relational decision trees along with some comprehensible descriptions of the learned subdivision of the given set of relational objects (Sect. 4).

[1] A combination of MATCHBOX with other relational decision tree learners like INDIGO ([10]) is possible.

2 Learning Relational Decision Trees Using TRITOP

TRITOP constructs a decision tree containing structural attributes (test graphs) from a training set with preclassified relational structures (example graphs). Both test graphs (attributes) and example graphs are represented by logical clauses. The attributes specify structured tests and are kept as small as possible for efficiency reasons. This means TRITOP learns a discriminating classifier and not a characteristic description of the concept. The theoretical foundations of TRITOP and a sketch of its tree building algorithm can be found in [9,11].

We will explain the peculiar features of TRITOP using the training set in Fig. 1 that contains seven blocks world configurations. For each configuration

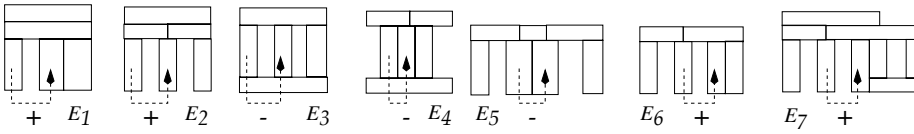


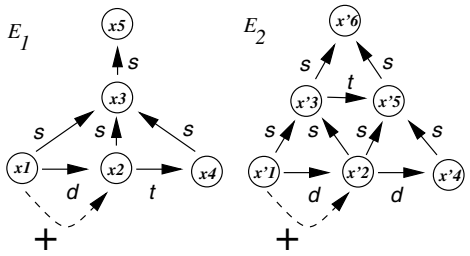
Fig. 1. The training setfor learning a binary relation. The dashed arrow indicates the pair of elementary objects to which the class is assigned

a pair of its blocks is classified as either "+" if they are the two supports of a simple arch, or as "−" if they are not. Especially, the two blocks have to support a third block, are not allowed to touch and are required to stand directly on the floor. For the first example, the class label "+" states that the ordered pair indicated by the arrow forms such a pair of supports.

Using the relations s for "supports", t for "touches" and d for "does not touch", blocks world situations can be described by labeled directed graphs, see Fig. 2. A classified graph can be represented by a definite clause – thus allowing more than binary relations – having the classification as its head and the description of the graph as its body. The classification consists of the classified nodes together with their class value. The nodes in the graphs are represented by **node variables**[2] in the clauses, i.e. variables of the distinguished sort *Node*. We allow the clauses to contain also variables of other sorts, e.g. real valued variables that have the sort *real*. The class constants "+" and "−" belong to the sort *Class*. Using this clausal representation, the examples E_1 and E_2 can be described by the clauses in Fig. 2 on the right. We have used the node variables (x_1, x_2) and (x_1', x_2') respectively to represent the classified blocks in E_1 and E_2 of Fig. 1.

TRITOP extracts test graphs from the training set and uses them to produce a classifier for graphs. A relational decision tree is a decision tree whose inner nodes are labeled with structural attributes, particularly test clauses describing

[2] For formal reasons, nodes are not represented by constants as in other ILP systems. This way, we can use the same representation for example and attribute clauses. Because of this representation, an example clause can already be seen as a classifier.



$class(x_1, x_2, +) \leftarrow$
$d(x_1, x_2), t(x_2, x_4), s(x_1, x_3),$
$s(x_2, x_3), s(x_4, x_3), s(x_3, x_5)$

$class(x'_1, x'_2, +) \leftarrow$
$d(x'_1, x'_2), d(x'_2, x'_4), s(x'_1, x'_3),$
$s(x'_2, x'_3), s(x'_2, x'_5), s(x'_4, x'_5),$
$t(x'_3, x'_5), s(x'_3, x'_6), s(x'_5, x'_6)$

Fig. 2. The graphs for the examples E_1 and E_2 in Fig. 1 and their clausal representations

the occurrence or lack of certain substructure respectively. The leaves of the tree are labeled with class values. A decision tree classifying the training set in Fig. 1 correctly is[3]

$$\begin{array}{l} class(y_1, y_2, y) \leftarrow d(y_1, y_2) \\ 0 \rightarrow - \\ 1 \rightarrow class(y_1, y_2, y) \leftarrow s(y_4, y_1) \\ \qquad 0 \rightarrow + \\ \qquad 1 \rightarrow - \end{array} \tag{1}$$

The classification of an example works as follows. E. g. pair (x_1, x_2) of E_1 is classified as $+$, because the first test attribute $A_1 = class(y_1, y_2, y) \leftarrow d(y_1, y_2)$ occurs for (x_1, x_2), but the second attribute $A_2 = class(y_1, y_2, y) \leftarrow s(y_4, y_1)$ does not. More formally speaking, the definite attribute clause A_1 **α-subsumes** the example clause E_1 ($A_1 \leq^{\alpha} E_1$). This means that A_1 occurs as a substructure of E_1 using the embedding alphabetical (injective) substitution $\theta = \{y_1 \leftarrow x_1, y_2 \leftarrow x_2, y \leftarrow +\}$, i.e. $A_1\theta \subseteq E_1$.

In contrast to A_1, the relation $A_2 \leq^{\alpha} E_1$ does not hold, because there is no substitution θ' for which $A_2\theta' \subseteq E_1$. Therefore E_1 is classified using the leaf labeled with $+$. It should be noted, that each attribute is allowed to consist of more than one literal. The bindings of the variables when evaluating a test do not influence the evaluation of the subsequent tests in the tree, i.e. the variables only have a local meaning. The variable y occurring in the attributes A_1 and A_2 is a dummy variable that is *not* used for determining the class of an example to be classified.

The information that both blocks support a third block – i.e. the literals $s(y_1, y_3)$, $s(y_2, y_3)$ – is not contained in A_1, because it is redundant with respect to the given small training set. Redundant means here, that any two blocks in the training set, that do not touch, always support a third block. Redundant literals yield no discrimination of the classes but increase the complexity of the tree. Thus, TRITOP avoids the inclusion of such literals though they have explanatory power. In the prototype construction step, literals are included that

[3] Each attribute is followed by the two possible outcomes of the test: non-occurrence (0) or occurrence (1) of the specified structure. The tree structure is reflected by the indentation.

are (conditionally) redundant with respect to the classification but that bear useful information about the learning domain.

Learning the described concept is an example for learning a *binary relation*. The example graphs must contain all relevant information, i.e. a description of the features of the nodes in the example tuple, together with their relations and their relational context. In addition to learning binary or even n-place relations, it is possible to learn a classifier for graphs with TRITOP. An example is given in Sect. 4: the classification of chemical compounds. In this case, the heads of the example and test clauses contain no node variables.

2.1 α-Subsumption

Using alphabetical substitutions for subsumption goes back on S. Vere ([24]). We use the term α-substitution instead of "alphabetical substitution", because α-substitutions need only be alphabetical on variables of the sort *Node*, but not on variables of other sorts, e.g. real valued variables occurring in constraints (for details see [9,11]). In TRITOP, the α-subsumption plays the role of an extended subgraph relation, though it can also be understood as a restricted form of logical implication.

The α-subsumption is a restriction of the more general θ-subsumption that uses general instead of alphabetical substitutions. To some respect, the concept of the α-subsumption captures the basic idea of *monomorphisms* that are well-known in graph theory. Besides Vere's work, the α-subsumption is also related to concepts found in [13,12,15,7].

The underlying assumption of α-subsumption, namely that different variable identifiers used in tests clauses denote different objects is very natural in many *practical* application domains. For example, in the mutagenicity domain (e.g. [22, 23]) the task is to learn a classifier for nitroaromatic and heteroaromatic chemical compounds predicting their mutagenicity. Highly mutagenic compounds can often cause cancer.

In the original version of the data the predicates *bond* and *atm* are used to define the properties of each compound in the training set. The description of compound *d191* for example contains the facts

atm(d191, d191_1, c, 22, -0.133*), atm(d191, d191_2, c, 22,* -0.133*),*
bond(d191, d191_1, d191_2, 7), bond(d191, d191_2, d191_3, 7), ...

which express that the compound *d191* has two atoms named *d191_1* and *d191_2* which are connected by an aromatic bond (bond type *7*). Both atoms are characterized to be carbon atoms (constant *c*) that have the partial charge *-0.133*. The atom type *22* specifies *d191_1* to be a special kind of aromatic carbon atom[4]. Each compound in the training set is classified as either belonging to class "active" (mutagenic) or "inactive" (non mutagenic).

[4] The atom types are specific to a system for molecular modeling (QUANTA).

A hypothesis generated by the learning system PROGOL contains the definite clause

$$active(A) \leftarrow bond(A,B,C,2),\ bond(A,D,B,1),\ atm(A,D,c,21,E)$$

(from [22]), where A represents the substance to be classified and B, C *and* D three of its atoms. The atoms B and C are connected by a double bond which is indicated by the number 2 in the first literal of the body of the clause. The atoms D and B are connected by a single bond which is stated in the second literal. Additionally the atom D is defined to be a type 21 carbon atom with an unspecified partial charge E.

When classifying a new substance using PROGOL's clause, the variables A, B, C, and D will be instantiated with different constants, though this is not explicitly stated in the clause, e.g. by the inclusion of constraints such as $B \neq C$ etc. The inclusion of explicit inequality constraints makes the clause more precise (possibly preventing classification errors) and is for example discussed in [14]. The inclusion of inequality literals leads to an extreme computational overhead, e.g. when evaluating clauses during hypothesis construction and when constructing the *lgg*. Therefore, in TRITOP the atoms of a compound are represented using node variables which cannot (by definition) be instantiated with the same value. The inequalities are handled implicitly by the α-subsumption algorithm.

Using α-subsumption has some nice effects: its usage leads to an effectively finite hypothesis space. Also, least general generalizations can be more easily be interpreted as compared with Plotkin's *lgg*, s. next section.

3 Construction of Prototypes

Prototypes can be used in decision trees in two different ways:

1. they can be used to **replace discriminating attributes by descriptive ones.** This is done for a test in the tree by computing the generalization of all objects in the training set that pass the 1-branch of the respective test, see Fig. 3.
2. The **prototypes** can be used to **explain the subsets** that correspond to the leaves of the tree.

In the second case, the decision tree algorithm performs a kind of "supervised clustering" of each class into subclasses. All examples reaching a leaf are known to share a number of common structural patterns which discriminate them from examples of the other class. For each leaf a subclass prototype can be constructed by computing the generalisation of the examples belonging to that leaf. This approach is sketched Fig. 4 and will be investigated in the following.

In our framework, prototypes correspond to least general generalizations that have to be computed with respect to α-subsumption. A clause C is said to be an **α-generalization** of C_1 and C_2, if $C \leq^{\alpha} C_1$ and $C \leq^{\alpha} C_2$ holds. C is a **least**

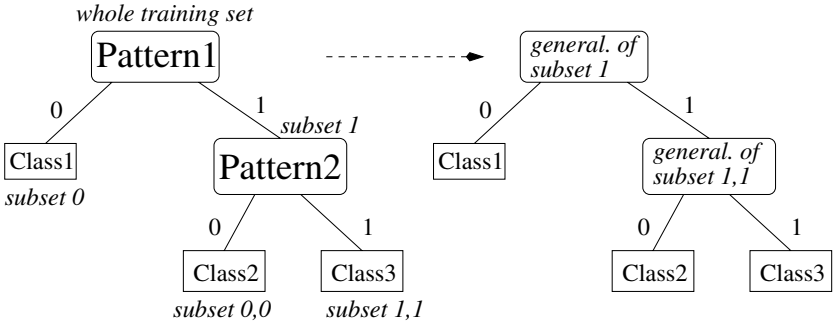


Fig. 3. Generation of a decision tree with augmented test graphs from a decision tree with discriminating tests.

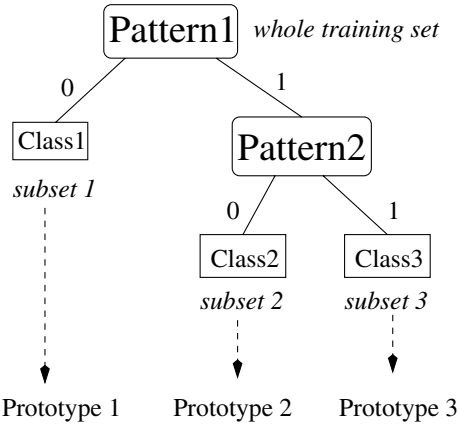
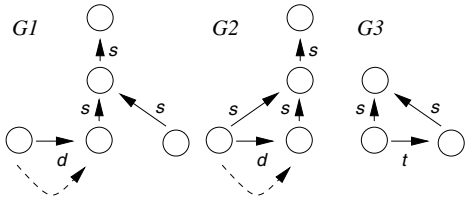


Fig. 4. Sketch of the prototype generation

general α-generalization, if for every other α-generalization $C_3 \not\in C$ the relation $C \leq^{\alpha} C_3$ does not hold. In contrast to Plotkin's *lgg* ([17]), there may be several α-*lgg*s that are not comparable with respect to the relation $\leq^{\alpha}$, but it can be shown that the *lgg* is composed of the α-*lgg*s ([9,15]). An α-generalisation is *shorter* than the original clauses while the *lgg* of two clauses C_1 and C_2 may posses $\|C_1\| \cdot \|C_2\|$ literals in the worst case. In Fig. 5 all α-*lgg*s of E_1 and E_2 (Fig. 2) are shown together with Plotkin's *lgg* (unreduced). The dashed arrow indicates which variables occur in the head of the clause corresponding to the graph. The third generalisation G_3 is not a definite clause. Hence it is not useful as a structural attribute and will be discarded by TRITOP's attribute construction algorithm, for details see [11].

It is obvious that each α-*lgg* clause in Fig. 5 characterizes a commonality of E_1 and E_2 in an intelligible way, while the *lgg* has no direct intuitive interpretation. Moreover, the clauses G_1 and G_2 in Fig. 5 themselves can be seen as **prototypical** descriptions of blocks world configurations which is not the case for the *lgg*.



G_1 : $class(y_{11}, y_{22}, y) \leftarrow d(y_{11}, y_{22}), s(y_{22}, y_{35}), s(y_{44}, y_{35}), s(y_{35}, y_{56})$

G_2 : $class(y_{11}, y_{22}, y) \leftarrow d(y_{11}, y_{22}), s(y_{11}, y_{33}), s(y_{22}, y_{33}), s(y_{33}, y_{56})$

G_3 : $\leftarrow s(y_{23}, y_{36}), s(y_{45}, y_{36}), t(y_{23}, y_{45})$

lgg: $class(y_{11}, y_{22}, y) \leftarrow d(y_{11}, y_{22})$, $d(y_{12}, y_{24})$, $s(y_{11}, y_{33})$, $s(y_{12}, y_{33})$, $s(y_{12}, y_{35})$, $s(y_{14}, y_{35})$, $s(y_{13}, y_{36})$, $s(y_{15}, y_{36})$, $s(y_{21}, y_{33})$, $s(y_{22}, y_{33})$, $s(y_{22}, y_{35})$, $s(y_{24}, y_{35})$, $s(y_{23}, y_{36})$, $s(y_{25}, y_{36})$, $s(y_{41}, y_{33})$, $s(y_{42}, y_{33})$, $s(y_{42}, y_{35})$, $s(y_{44}, y_{35})$, $s(y_{43}, y_{36})$, $s(y_{45}, y_{36})$, $s(y_{31}, y_{53})$, $s(y_{32}, y_{53})$, $s(y_{32}, y_{55})$, $s(y_{34}, y_{55})$, $s(y_{33}, y_{56})$, $s(y_{35}, y_{56})$, $t(y_{23}, y_{45})$

Fig. 5. The α-*lgg*s G_1, G_2 and G_3 of E_1 and E_2, and $lgg(E_1, E_2)$ with $y_{ij} = lgg(x_i, x'_j)$

Constructing α-*lgg*s is computationally extremely expensive for large graphs. This is shown in [9] based on the results in [12,15,14]. Therefore, a connectionist approach has to be used for prototype construction that will be described in the following.

3.1 A Connectionist Approach for Structural Matching

For the connectionist prototype construction, the objects of the training set are represented as labeled graphs, i.e. all node variables represent nodes, and predicates are transformed into labels of nodes and edges. Thus, all features are bound to objects or relations between objects, and all features of a node or relation (edge), respectively, are handled as a single feature vector.

As is described above, TRITOP constructs decision trees where every leaf corresponds to a subset of the training set. The aim of the connectionist reconstruction of prototypes is to find a common substructure of all training examples belonging to a leaf of the learned tree. The path from the root of the tree to a leaf consists of a sequence of structural attributes together with a value being either 0 or 1. If in this sequence all attributes are removed that have the outcome 0, the remaining attributes describe small substructures contained in all examples belonging to the leaf.

During prototype generation, a maximal common substructure of all examples belonging to the respective leaf is found which contains the small substructures given by TRITOP's attributes. Computing an α-*lgg* is transformed into the problem of finding an optimal match between the nodes of the graphs, which in general belongs to the class of NP-complete problems. The match is determined by introducing a user defined threshold for a minimum similarity of nodes which can be mapped onto each other. Therefore a similarity of node and edge labels has to be defined. In the mutagenesis example, the similarity between atoms can be defined according to the chemical properties of their elements, charges etc. The set of admissible matches is restricted to the set of injective mappings.

If there exist several least general generalizations of the examples in a subset (like in Fig. 5), then it is possible to provide either *all* of the prototypes, or to generate a single, best generalization that captures the most interesting commonalities of the examples. The latter alternative is chosen by the system MATCHBOX.

The best match between graphs can be described by a quadratic objective function, which rewards matches between similar nodes and edges according to their similarity and assigns a penalty to inconsistencies in the match. A detailed overview on the objective functions used in MATCHBOX is given in [20,21].

It has been shown that such quadratic optimization tasks can be solved by Hopfield-like Artificial Neural Nets, i.e. bidirectional associative memories. For a quadratic objective function, a neural net can be constructed so that the energy of this net is described by the objective function. Then the dynamics of the net ensures that the net moves from high energy states to states with lower energy. Under certain conditions, the net converges to a steady state that corresponds to a local minimum of the energy function, i.e. to a solution of the optimization task. Despite of the criticism of Hopfield nets as a means of solving NP-complete problems ([3,4]), successful applications show that those nets can provide good and fast solutions for many practical problems ([19]).

A special neural net from the Hopfield family based on the approach described in [8,25] has been developed and investigated theoretically in [20]. On the basis of these results, the system MATCHBOX (for details, see [21]) has been developed. MATCHBOX performs the matching of labeled graphs, using a knowledge base of types of node and edge labels and the similarity measures defined for these types of labels. Labels can be feature vectors with components of different types, for instance real numbers, ordered sets, symbols or elements of a taxonomic (IS-A-) hierarchy. As a result of a graph match, a quantitative estimation of the similarity of the two graphs and the generalised common subgraph is returned.

In order to demonstrate how the neural approach works, in Fig. 6 an example is given. The figure shows the neural net which is produced from the graphs E_1 and E_2 from Fig. 2. Every unit of the net describes the mapping between two nodes. The nodes forming the first and second argument of the class predicate in E_1 can only be mapped onto their corresponding nodes in E_2. The units of the net have connections with negative weights (dashed lines in the figure), if they violate the injectivity constraint. The unit $[x3|x'3]$ for instance is connected by a negative connection to $[x3|x'4]$, because the node $x3$ cannot be mapped onto $x'3$ and $x'4$ at the same time. Connections with positive weights are drawn between units which correspond to node matches where the relations between the nodes are preserved in the mapping. So the units $[x1|x'1]$ and $[x2|x'2]$ are connected by a positive link because the relation between $x1$ and $x2$ and the one between $x'1$ and $x'2$ are identical. After the construction of the net, the units are given an initial activation and after some steps of propagation, the net settles at a steady state. The active units in this state form a injective mapping between nodes of the two graphs. A steady state of the net corresponding to a solution of the matching problem consists of some active nodes which are connected by positive,

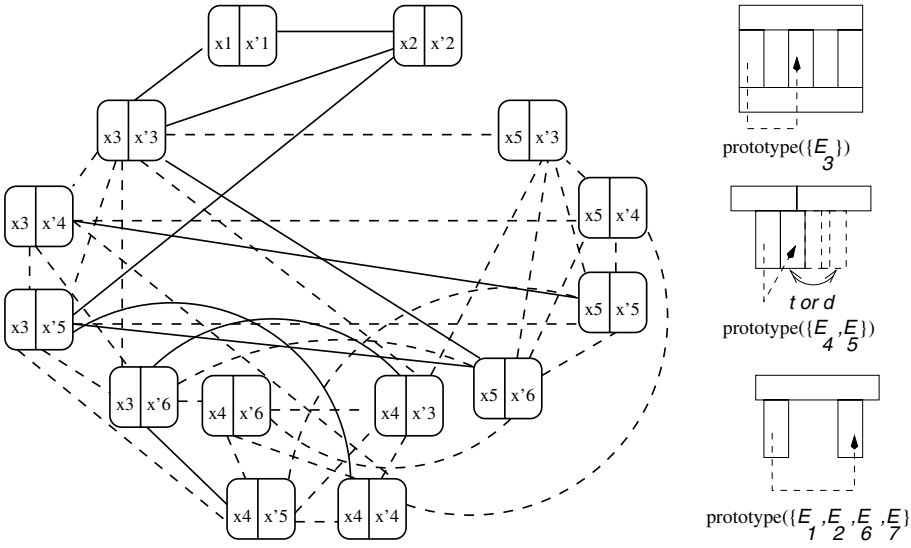


Fig. 6. (a) The neural net for E_1 and E_2 (b) prototypes for the tree in (1)

but not by negative links while the rest of the units is inactive. Steady states in the example, for instance, consist of the nodes $[x1|x'1], [x2|x'2], [x3|x'3], [x5|x'6]$ or $[x1|x'1], [x2|x'2], [x3|x'5], [x4|x'4], [x5|x'6]$. These mappings correspond to the first and second α-*lgg* in Fig. 5.

In the prototype construction step for a leaf of the decision tree, MATCHBOX receives as its input all training examples of a leaf. MATCHBOX then computes a generalised common subgraph of all the examples corresponding to the leaf. Take, for instance, the above decision tree (1). This tree divides the examples from Fig. 1 in three subsets: $\{E_3\}$ (negative example, contains a bar that supports the columns of the thing that otherwise would be an arch), $\{E_4, E_5\}$ (negative examples, the columns touch each other) and $\{E_1, E_2, E_6, E_7\}$ (positive examples). MATCHBOX returns the prototypes shown in Fig. 6. Especially the prototype for the positive examples $\{E_1, E_2, E_6, E_7\}$ provides a characterization of the concept that is much better understandable than the original tree in (1). It is a generalization of the pairwise α-*lgg* G_2 depicted in Fig. 5. The "prototype" for subset $\{E_3\}$ coincides with the single example in the subset. The prototype for the subset $\{E_4, E_5\}$ can be seen as a generalized typical counterexample for class "+", because class "−" is assumed to contain the complement of class "+"[5].

[5] Depending on the task, it may not be possible to construct reasonable prototypes for the negative class.

4 The Mutagenesis Dataset

In this section, we will present results for our approach on the mutagenicity dataset introduced in Sect. 2. The data described e.g. in [22,23] consists of two datasets of chemical compounds. The first data set contains 188 compounds that are "regression friendly", i.e. their mutagenicity can be predicted using standard regression methods. The second dataset contains 42 regression "unfriendly" compounds. It turned out that – with respect to accuracy – TRITOP performs very well on the two datasets when compared to other learning systems, see [11].

For our experiments with TRITOP/MATCHBOX we used a transformed version of the dataset. For example, the substance *d191* containing the literals *atm(d191, d191_1, c, 22, -0.133)*, *atm(d191, d191_2, c, 22, -0.133)*, *bond(d191, d191_1, d191_2, 7)*, and *bond(d191, d191_2, d191_3, 7)* is characterized by the transformed example clause

$$\begin{aligned} \textit{class(inactive)} \leftarrow\ & \textit{charge(D191\_1, -0.133), type22(D191\_1), c(D191\_1),} \\ & \textit{charge(D191\_2, -0.133), type22(D191\_2), c(D191\_2),} \\ & \textit{seven(D191\_2, D191\_1), seven(D191\_1, D191\_2) ...} \end{aligned}$$

The node variable *D191_1* is used to represent the atom *d191_1*. We used the unary relations *c(D191_1)* and *type22(D191_1)* to characterize the atom's element and its type. The aromatic bond between *d191_1* and *d191_2* is expressed by the literals *seven(D191_2, D191_1)* and *seven(D191_1, D191_2)*. In general, relations that have more than 2 places have to be replaced by relations with an arity of at most 2.

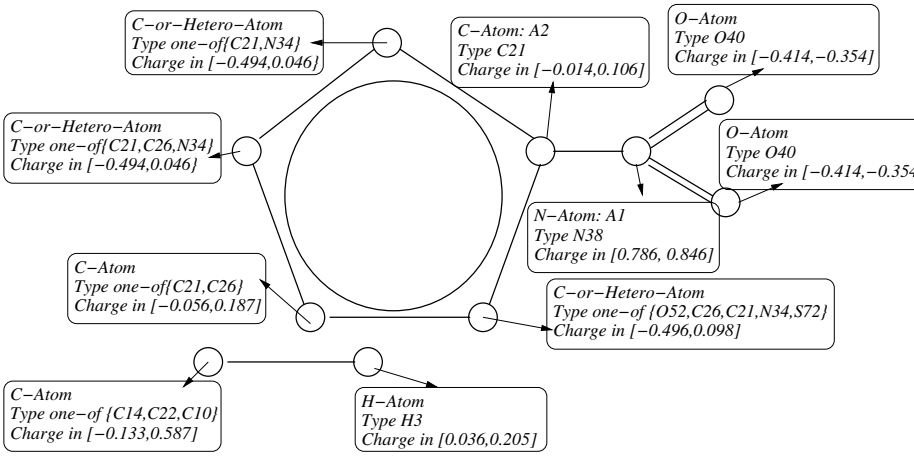


Fig. 7. The prototype for the subclass of mutagenic compounds

For the smaller mutagenesis dataset, TRITOP constructs the tree

$$\begin{array}{l}(class(y) \leftarrow one(A1, A2), type21(A2), n(A1)) \\ 0 \rightarrow - \\ 1 \rightarrow +\end{array} \tag{2}$$

with an attribute that discriminates between the classes active and inactive (*one* denotes a single bond). For a chemist who wants to asses chemical validity of the learned hypothesis this would be unsatisfying. So we used MATCHBOX to construct prototypes. In the 42 chemicals dataset, the compounds *f1*, *f5*, *f6*, *d191*, *f2*, *f4*, *d197*, and *f3* are classified using the first leaf in the above tree. MATCHBOX constructed the prototype in Fig. 7. The prototype found by MATCHBOX contains the NO_2 functional group and an aromatic 5-ring with heteroatoms like oxygen, sulfur and nitrogen in certain positions, an additional Atom-Bond-Atom-triple and constraints for the types of atoms and their charges. Obviously, this description is much more comprehensive and much easier to understand than the original decision tree and PROGOL's clause for this data set given in Sect. 2.

For reasons of space, we do not report the decision tree for the larger mutagenesis datatset. For the construction of prototypes, we used a decision tree with 29 subclasses. The decision tree was constructed by TRITOP on the whole training set of 188 examples, using parameter settings that turned out to produce a small estimated error using 10-fold cross validation.

A rather large prototype computed for a subset of 5 active compounds corresponding to a leaf of this tree can be found in Fig. 8. The compound contains several rings together with a NO2 group. The NO2 group fails to be connected to the rest of the prototype. A look at the examples suggests that the group should be connected to either atom $x12$ or $x1$. The problem of the missing edge arises because of the symmetries in the dataset. Ongoing research on graph matching with WTA nets is concerned with handling such problems.

A connected prototype for 9 *inactive* examples is shown in Fig. 9. In contrast to the blocks world example, where the negative class "−" is defined by the absence of structures only, it may be the case that there are substructures whose presence leads to an inactivity of the compound. It should be noted, that a system like FOIL or PROGOL is in general only able to represent complex patterns for the class *active* (expressed as clauses). Structural features of the class *inactive* can only be expressed by using *single* negated literals occurring in the clauses for class *active*.

5 Conclusions

In this paper, a hybrid approach to relational learning has been introduced which combines the advantages of decision tree classifiers and class descriptions by prototypes. In a first step, the learning algorithm TRITOP constructs a relational decision tree. Due to the sophisticated selection and optimisation of the structural attributes used as tests of the resulting decision trees, the trees built

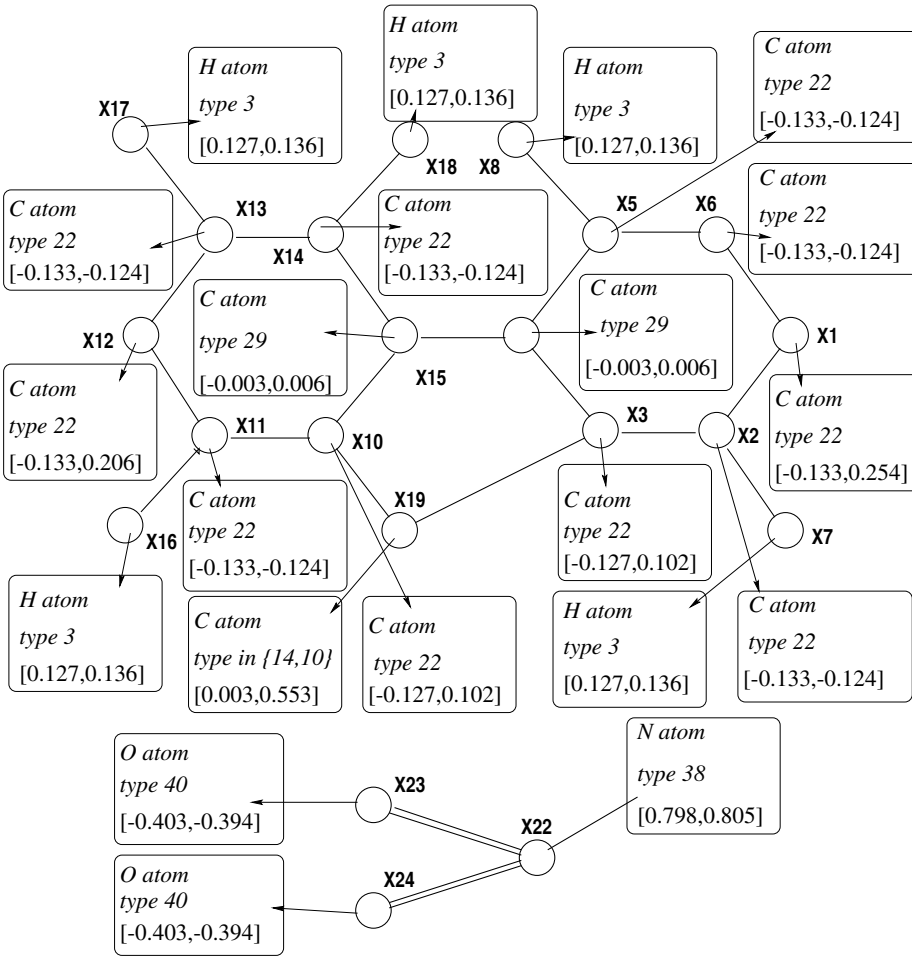


Fig. 8. Prototype for a subset of 5 active compounds in the 188 examples dataset

by TRITOP show very high classification accuracy. In order to provide the user with an understandable, concise description of the results of the tree learning and construction step, in a second step the subdivision of the set of training examples given by tests of the tree is used to produce a set of prototypes. Each prototype represents a subclass of examples produced by a leaf of the tree. MATCHBOX is a connectionist, knowledge based system for graph matching and generalised prototype construction. With MATCHBOX, prototypes for sets of examples are computed efficiently. Prototypes built by MATCHBOX are generalised common substructures of the structures in the set which include constraints on the values of e.g. real-valued variables. Thus, TRITOP/MATCHBOX not only provides a powerful tool for learning both discriminations and descriptions of classes of relational objects, including features like handling of noisy data and learning

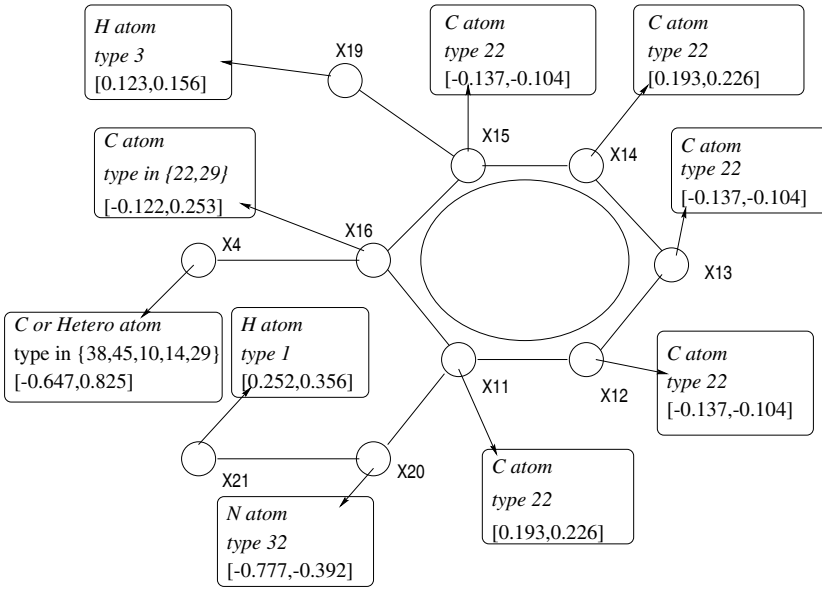


Fig. 9. Prototype for inactive compounds

constraints for real valued variables. It also contributes to the solution of the general problem of the high complexity of learning in the domain of relational descriptions.

Future work will include experimental evaluation on other datasets. We also want to improve the performance of the MATCHBOX algorithm when applied to large graphs or graphs containing symmetries.

An interesting point that will be investigated in future research is the problem of negation. The constructed class prototypes only provide positive structural information about the subclasses, whereas the decision tree is able to characterize subclasses by the absence of patterns. I.e. the prototypes in general do not bear the full information of the decision tree. The relationship of the prototypes and the tree have to be investigated in more detail.

Acknowledgments. We thank Andrea Doliana who has computed the prototype for the 188 examples data set during his master's thesis. We thank Brijnesh-Johannes Jain for co-supervising the thesis.

References

1. G. Bisson. Conceptual clustering in a first order logic representation. In B. Neumann, editor, *Proc. of the 10th ECAI*, pages 458–462. John Wiley & Sons, 1992.
2. H. Blockeel, L. DeRaedt, and J. Ramon. Top-down induction of clustering trees. In J. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning*, pages 55–63, 1998.

3. J. Bruck and J. W. Goodman. On the power of neural networks for solving hard problems. *Journal of Complexity*, 6:129–135, 1990.
4. L.I. Burke and J.P. Ignizio. Neural networks and operations research: An overview. *Computers Ops. Res.*, 19(3/4):179–189, 1992.
5. B. Dolsak and S. Muggleton. The application of Inductive Logic Programming to finite element mesh design. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, London, 1992.
6. W. Emde and D. Wettschereck. Relational instance based learning. In L. Saitta, editor, *Proc. 13th ICML*, pages 122–130. Morgan Kaufmann Publishers, 1996.
7. F. Esposito, A. Laterza, D. Malerba, and G. Semerano. Refinement of datalog programs. In *ICML '96 Workshop on "Datamining with Inductive Logic Programming*, pages 73–94, 1996.
8. J. A. Feldman, M. A. Fanty, N. H. Goddard, and K. J. Lynne. Computing with structured connectionist networks. *CACM*, 31(2):170–187, February 1988.
9. P. Geibel. *Induktive Konstruktion von merkmalsbasierten und logischen Klassifikatoren für relationale Strukturen*. PhD thesis, TU Berlin, 1999.
10. P. Geibel and F. Wysotzki. Learning relational concepts with decision trees. In L. Saitta, editor, *Machine Learning: Proc. of the 13th Int. Conf.* Morgan Kaufmann Publishers, San Fransisco, CA, 1996.
11. P. Geibel and F. Wysotzki. A logical framework for graph theoretic decision tree learning. In N. Lavrac and S. Dzeroski, editors, *Proc. ILP 97*. Springer, 1997.
12. D. Haussler. Learning Conjunctive Concepts in Structural Domains. *Machine Learning*, 4:7–40, 1989.
13. F. Hayes-Roth and J. McDermott. Knowledge acquisition from structural descriptions. In Raj Reddy, editor, *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 356–362, Cambridge, MA, August 1977. William Kaufmann.
14. N. Helft. Inductive generalization: A logical framework. In *Proceedings of the Second Working Session on Learning*, pages 149–157, 1987.
15. J.-U. Kietz. A comparative study of structural most specific generalisations used in machine learning. In *Proc. ILP-93*, pages 149–164, Ljubljana, Slovenia, 1993. J. Stefan Institute Tech. Rep. IJS-DP-6707.
16. M. Manago. Knowledge intensive induction. In A. M. Segre, editor, *Proceedings of the 6th International Workshop on Machine Learning*, pages 151–155, Ithaca, 1989. Morgan Kaufmann.
17. G. D. Plotkin. A note on inductive generalization. In *Machine Intelligence*, pages 153–164. Edinburgh University Press, 1969.
18. L. De Raedt and H. Blockeel. Using logical decision trees for clustering. In N. Lavrač and S. Džeroski, editors, *Proc. of the 7th Int. WS on ILP*, volume 1297 of *LNAI*, pages 133–140. Springer, September17–20, 1997.
19. K. Schädler and F.Wysotzki. A connectionist approach to distance-based analysis of relational data. In X. Liu, P. Cohen, and M. Berthold, editors, *Proc. of the IDA-97*, pages 137–148. Springer, 1997.
20. K. Schädler and F. Wysotzki. Theoretical foundations of a special neural net approach for graphmatching. Technical Report 96–26, TU Berlin, CS Dept., 1996.
21. K. Schädler and F. Wysotzki. Comparing structures using a hopfield-style neural network. *Applied Intelligence – special issue on Neural Networks and Structured Knowledge*, 11(1):5–30, 1999.
22. A. Srinivasan, S. Muggleton, R. King, and M. Sternberg. Mutagenesis: Ilp experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proc. of the Fourth Intl. WS on ILP*, number 237 in GMD-Studien, pages 217–232, 1994.

23. A. Srinivasas, R.D. Kind, S.H. Muggleton, and M. J. E Sternberg. Carcinogenesis predictions using ILP. In N. Lavrac and S. Dzeroski, editors, *Proc. ILP-97*, number 1297 in LNAI, pages 273–287. Springer-Verlag, 1997.
24. S. A. Vere. Induction of Concepts in the Predicate Calculus. In *Proc. of the Fourth Intl. Joint Conf. on AI*, volume 1, pages 281–287, 1975.
25. F. Wysotzki. Artificial Intelligence and Artificial Neural Nets. In *Proc. 1st WS on AI*, Shanghai, September 1990. TU Berlin and Jiao Tong University Shanghai.

The Well-Founded Semantics Is a Stratified Fitting Semantics

Pascal Hitzler and Matthias Wendt

Artificial Intelligence Institute, Department of Computer Science
Dresden University of Technology, Dresden, Germany
{phitzler,mw177754}@inf.tu-dresden.de

Abstract. Part of the theory of logic programming and nonmonotonic reasoning concerns the study of fixed-point semantics for these paradigms. While several different semantics have been proposed, and some have been more successful than others, the exact relationships between the approaches have not yet been fully understood. In this paper, we give new characterizations, using level mappings, of the Fitting semantics, the well-founded semantics, and the weakly perfect model semantics. The results will unmask the well-founded semantics as a stratified version of the Fitting semantics.

1 Introduction

One of the very stimulating research questions in logic programming and nonmonotonic reasoning has been the search for an appropriate declarative understanding of negation. Several different semantics for logic programs have been proposed, see e.g. [1], each being more or less convincing, depending on one's point of view, which may be that of a programmer, or motivated by commonsense reasoning.

Among the semantics based on three-valued logics, the Fitting semantics [2] and the well-founded semantics [3] are prominent and widely acknowledged choices. Theoretical relationships between them have been etablished, e.g. in [4] by using lattice-based logic programming in four-valued logic, an approach which was recently extended in [5]. The development of the weakly perfect model semantics, due to [6], was motivated by the intuition that recursion should be allowed through positive information, but not through negation. As such, it was developed out of the notion of stratification [7,8]. We will see that the well-founded semantics achieves this very goal in a much better way than the weakly perfect model semantics. This will probably be no surprise for the specialist, although we are not aware of any formal argument for this case, and we believe that a presentation from our point of view is rather clear and convincing.

We will provide new, and uniform, characterizations, based on level mappings, of the Fitting semantics, the well-founded semantics, and the weakly perfect model semantics. Level mappings are mappings from Herbrand bases to ordinals, i.e. they induce orderings on the set of all ground atoms while disallowing infinite descending chains. They have been studied in termination analysis

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 205–221, 2002.

for logic programming, e.g. in [9], where they appear naturally, they have been used for defining classes of programs with desirable semantic properties, e.g. in [7,8], they are intertwined with topological investigations of fixed-point semantics in logic programming, as studied e.g. in [10,11,12], and are relevant to some aspects of the study of relationships between logic programming and artificial neural networks [13]. Our motivation, however, is quite different. We will employ level mappings in order to give uniform characterizations of different semantics in nonmonotonic reasoning, and we will this way employ them to unmask the well-founded semantics as a stratified version of the Fitting semantics, i.e. we will see that the difference between the Fitting semantics and the well-founded semantics is, in a nutshell, that at some particular point the former prevents recursion, while the latter at the same point prevents only recursion through negation.

As a trivial corollary of our results, we obtain that the Fitting semantics is more skeptical than the weakly perfect model semantics, which is in turn more skeptical than the well-founded semantics (Corollary 4).

The paper is structured as follows. Section 2 contains preliminaries which are needed to make the paper relatively self-contained. Section 3 contains our new characterization of the Fitting semantics, while Sect. 4 covers the new characterization of the well-founded semantics. In Sect. 5 the weakly perfect model semantics will be studied. We conclude with a summary and a discussion of further work in Sect. 6.

Acknowledgement. We thank three anonymous referees for their comments, which helped to improve the presentation.

2 Preliminaries and Notation

A (*normal*) *logic program* is a finite set of (universally quantified) *clauses* of the form $\forall(A \leftarrow A_1 \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_m)$, commonly written as $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$, where A, A_i, and B_j, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, are atoms over some given first order language. A is called the *head* of the clause, while the remaining atoms make up the *body* of the clause, and depending on context, a body of a clause will be a set of literals (i.e. atoms or negated atoms) or the conjunction of these literals. Care will be taken that this identification does not cause confusion. We allow a body, i.e. a conjunction, to be empty, in which case it always evaluates to true. A clause with empty body is called a *unit clause* or a *fact*. A clause is called *definite*, if it contains no negation symbol. A program is called *definite* if it consists only of definite clauses. We will usually denote atoms with A or B, and literals, which may be atoms or negated atoms, by L or K.

Given a logic program P, we can extract from it the components of a first order language. The corresponding set of ground atoms, i.e. the *Herbrand base* of the program, will be denoted by B_P. For a subset $I \in B_P$, we set $\neg I = \{\neg A \mid A \in B_P\}$. The set of all ground instances of P with respect to B_P will be denoted

by $\mathsf{ground}(P)$. A (*three-valued* or *partial*) *interpretation* I for P is a subset of $B_P \cup \neg B_P$ which is *consistent*, i.e. for each $A \in B_P$ at most one of A and $\neg A$ is contained in I. We say that A is *true with respect to* (or *in*) I if $A \in I$, we say that A is *false with respect to* (or *in*) I if $\neg A \in I$, and if neither is the case, we say that A is *undefined with respect to* (or *in*) I. A body, i.e. a conjunction of literals, is true in an interpretation I if every literal in the body is true in I, it is false in I if one of its literals is false in I, and otherwise it is undefined in I. For a negated literal $L = \neg A$ we will find it convenient to write $\neg L \in I$ if $A \in I$. By I_P we denote the set of all (three-valued) interpretations of P. It is a complete partial order (cpo) via set-inclusion, i.e. it contains the empty set as least element, and every ascending chain has a supremum, namely its union. A *model* of P is an interpretation $I \in I_P$ such that for each clause $A \leftarrow \mathtt{body}$ we have that $\mathtt{body} \subseteq I$ implies $A \in I$. A *total* interpretation is an interpretation I such that no $A \in B_P$ is undefined in I.

For an interpretation I and a program P, an *I-partial level mapping* for P is a partial mapping $l : B_P \to \alpha$ with domain $\mathsf{dom}(l) = \{A \mid A \in I \text{ or } \neg A \in I\}$, where α is some (countable) ordinal. We extend every level mapping to literals by setting $l(\neg A) = l(A)$ for all $A \in \mathsf{dom}(l)$. A (*total*) *level mapping* is a total mapping $l : B_P \to \alpha$ for some (countable) ordinal α.

Given a normal logic program P and some $I \in I_P$, we say that $U \subseteq B_P$ is an *unfounded set* (*of P*) *with respect to I* if each atom $A \in U$ satisfies the following condition: For each clause $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ (at least) one of the following holds.

(Ui) Some (positive or negative) literal in $\mathtt{body}$ is false in I.
(Uii) Some (non-negated) atom in $\mathtt{body}$ occurs in U.

Given a normal logic program P, we define the following operators on I_P. $T_P(I)$ is the set of all $A \in B_P$ such that there exists a clause $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ such that $\mathtt{body}$ is true in I. $F_P(I)$ is the set of all $A \in B_P$ such that for all clauses $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ we have that $\mathtt{body}$ is false in I. $U_P(I)$ is the greatest unfounded set (of P) with respect to I, which always exists due to [3]. Finally, define

$$\Phi_P(I) = T_P(I) \cup \neg F_P(I) \qquad \text{and} \qquad W_P(I) = T_P(I) \cup \neg U_P(I)$$

for all $I \in I_P$. The operator Φ_P is due to [2], while W_P is due to [3]. Both are monotonic on the cpo I_P, hence have a least fixed point by the Tarski fixed-point theorem, and we can obtain these fixed points by defining, for each monotonic operator F, that $F \uparrow 0 = \emptyset$, $F \uparrow (\alpha + 1) = F(F \uparrow \alpha)$ for any ordinal α, and $F \uparrow \beta = \bigcup_{\gamma < \beta} F \uparrow \gamma$ for any limit ordinal β, and the least fixed point of F is obtained as $F \uparrow \alpha$ for some ordinal α. The least fixed point of Φ_P is called the *Kripke-Kleene model* or *Fitting model* of P, determining the *Fitting semantics* of P, while the least fixed point of W_P is called the *well-founded model* of P, giving the *well-founded semantics* of P.

Given a program P, we define the operator T_P^+ on subsets of B_P by $T_P^+(I) = T_P(I \cup \neg(B_P \setminus I))$. It is well-known that for definite programs this operator

is monotonic on the set of all subsets of B_P, with respect to subset inclusion. Indeed it is Scott-continuous [11] and, via Kleene's fixed-point theorem, achieves its least fixed point M as the supremum of the iterates $T_P^+ \uparrow n$ for $n \in \mathbb{N}$. So M is *the least two-valued model* of P. In turn, we can identify M with the total interpretation $M \cup \neg(B_P \setminus M)$, which we will call the *definite* (*partial*) *model* of P.

In order to avoid confusion, we will use the following terminology: the notion of *interpretation* will by default denote consistent subsets of $B_P \cup \neg B_P$, i.e. interpretations in three-valued logic. We will sometimes emphasize this point by using the notion *partial interpretation*. By *two-valued interpretations* we mean subsets of B_P. Both interpretations and two-valued interpretations are ordered by subset inclusion. Each two-valued interpretation I can be identified with the partial interpretation $I' = I \cup \neg(B_P \setminus I)$. Note however, that in this case I' is always a maximal element in the ordering for partial interpretations, while I is in general not maximal as a two-valued interpretation.

3 Fitting Semantics

We give a new characterization, using level mappings, of the Fitting semantics.

Definition 1. *Let P be a normal logic program, I be a model of P, and l be an I-partial level mapping for P. We say that P* satisfies (F) with respect to I and l, *if each $A \in \mathsf{dom}(l)$ satisfies one of the following conditions.*

- *(Fi)* $A \in I$ *and there exists a clause* $A \leftarrow L_1, \ldots, L_n$ *contained in* $\mathsf{ground}(P)$ *such that* $L_i \in I$ *and* $l(A) > l(L_i)$ *for all* i.
- *(Fii)* $\neg A \in I$ *and for each clause* $A \leftarrow L_1, \ldots, L_n$ *contained in* $\mathsf{ground}(P)$ *there exists* i *with* $\neg L_i \in I$ *and* $l(A) > l(L_i)$.

If $A \in \mathsf{dom}(l)$ satisfies (Fi), then we say that A satisfies (Fi) with respect to I and l, *and similarly if $A \in \mathsf{dom}(l)$ satisfies (Fii).*

Theorem 1. *Let P be a normal logic program with Fitting model M. Then M is the greatest model among all models I, for which there exists an I-partial level mapping l for P such that P satisfies (F) with respect to I and l.*

Proof. Let M_P be the Fitting model of P and define the M_P-partial level mapping l_P as follows: $l_P(A) = \alpha$, where α is the least ordinal such that A is not undefined in $\Phi_P \uparrow (\alpha + 1)$. The proof will be established by showing the following facts: (1) P satisfies (F) with respect to M_P and l_P. (2) If I is a model of P and l an I-partial level mapping such that P satisfies (F) with respect to I and l, then $I \subseteq M_P$.

(1) Let $A \in \mathsf{dom}(l_P)$ and $l_P(A) = \alpha$. We consider two cases.

(Case i) If $A \in M_P$, then $A \in T_P(\Phi_P \uparrow \alpha)$, hence there exists a clause $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ such that $\mathtt{body}$ is true in $\Phi_P \uparrow \alpha$. Thus, for all $L_i \in \mathtt{body}$ we have that $L_i \in \Phi_P \uparrow \alpha$, and hence $l_P(L_i) < \alpha$ and $L_i \in M_P$ for all i. Consequently, A satisfies (Fi) with respect to M_P and l_P.

(Case ii) If $\neg A \in M_P$, then $A \in F_P(\Phi_P \uparrow \alpha)$, hence for all clauses $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ there exists $L \in \mathtt{body}$ with $\neg L \in \Phi_P \uparrow \alpha$ and $l_P(L) < \alpha$, hence $\neg L \in M_P$. Consequently, A satisfies (Fii) with respect to M_P and l_P, and we have established that fact (1) holds.

(2) We show via transfinite induction on $\alpha = l(A)$, that whenever $A \in I$ (respectively, $\neg A \in I$), then $A \in \Phi_P \uparrow (\alpha + 1)$ (respectively, $\neg A \in \Phi_P \uparrow (\alpha + 1)$). For the base case, note that if $l(A) = 0$, then $A \in I$ implies that A occurs as the head of a fact in $\mathsf{ground}(P)$, hence $A \in \Phi_P \uparrow 1$, and $\neg A \in I$ implies that there is no clause with head A in $\mathsf{ground}(P)$, hence $\neg A \in \Phi_P \uparrow 1$. So assume now that the induction hypothesis holds for all $B \in B_P$ with $l(B) < \alpha$. We consider two cases.

(Case i) If $A \in I$, then it satisfies (Fi) with respect to I and l. Hence there is a clause $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ such that $\mathtt{body} \subseteq I$ and $l(K) < \alpha$ for all $K \in \mathtt{body}$. Hence $\mathtt{body} \subseteq M_P$ by induction hypothesis, and since M_P is a model of P we obtain $A \in M_P$.

(Case ii) If $\neg A \in I$, then A satisfies (Fii) with respect to I and l. Hence for all clauses $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ we have that there is $K \in \mathtt{body}$ with $\neg K \in I$ and $l(K) < \alpha$. Hence for all these K we have $\neg K \in M_P$ by induction hypothesis, and consequently for all clauses $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ we obtain that $\mathtt{body}$ is false in M_P. Since $M_P = \Phi_P(M_P)$ is a fixed point of the Φ_P-operator, we obtain $\neg A \in M_P$. This establishes fact (2) and concludes the proof.

We will use the following running example for illustration.

Example 1. The program consisting of the six rules

$$\begin{array}{lll} a \leftarrow \neg b & b \leftarrow c, \neg d & d \leftarrow e \\ b \leftarrow c, \neg a & c \leftarrow b, \neg e & e \leftarrow d \end{array}$$

has Fitting model $\emptyset$.

The following corollary follows immediately as a special case of Theorem 1, and is closely related to a result reported in [12].

Corollary 1. *A normal logic program P has a total Fitting model if and only if there is a total model I of P and a (total) level mapping l for P such that P satisfies (F) with respect to I and l.*

4 Well-Founded Semantics

We will provide a new characterization, via level mappings, of the well-founded semantics, and we will argue that from this new point of view the well-founded semantics can be understood as a stratified version of the Fitting semantics.

Let us first recall the definition of a (locally) stratified program, due to [7, 8]: A normal logic program is called *locally stratified* if there exists a (total) level mapping $l : B_P \to \alpha$, for some ordinal α, such that for each clause $A \leftarrow$

$A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ in $\mathsf{ground}(P)$ we have that $l(A) \geq l(A_i)$ and $l(A) > l(B_j)$ for all $i = 1, \ldots, n$ and $j = 1, \ldots, m$.

The notion of (locally) stratifed program was developed with the idea of preventing *recursion through negation*, while allowing recursion through positive dependencies. (Locally) stratified programs have total well-founded models.

There exist locally stratified programs which do not have a total Fitting semantics and vice versa. Indeed, the program consisting of the single clause $p \leftarrow p$ is locally stratified but p remains undefined in the Fitting semantics. Conversely, the program consisting of the two clauses $q \leftarrow$ and $q \leftarrow \neg q$ is not locally stratified but its Fitting model assigns to q the truth value *true*.

By comparing Definition 1 with the definition of locally stratified programs above, we notice that condition (Fii) requires a strict decrease of level between the head and a literal in the rule, independent of this literal being positive or negative. But, on the other hand, condition (Fii) imposes no further restrictions on the remaining body literals, while the notion of local stratification does. These considerations motivate the substitution of condition (Fii) by the condition (WFii), as given in the following definition.

Definition 2. *Let P be a normal logic program, I be a model of P, and l be an I-partial level mapping for P. We say that P* satisfies (WF) with respect to I and l, *if each $A \in \mathsf{dom}(l)$ satisfies one of the following conditions.*

(WFi) *$A \in I$ and there exists a clause $A \leftarrow L_1, \ldots, L_n$ contained in $\mathsf{ground}(P)$ such that $L_i \in I$ and $l(A) > l(L_i)$ for all i.*

(WFii) *$\neg A \in I$ and for each clause $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ contained in $\mathsf{ground}(P)$ (at least) one of the following conditions holds:*
(WFiia) There exists i with $\neg A_i \in I$ and $l(A) \geq l(A_i)$.
(WFiib) There exists j with $B_j \in I$ and $l(A) > l(B_j)$.

If $A \in \mathsf{dom}(l)$ satisfies (WFi), then we say that A satisfies (WFi) with respect to I and l, *and similarly if $A \in \mathsf{dom}(l)$ satisfies (WFii).*

We note that conditions (Fi) and (WFi) are identical. Indeed, replacing (WFi) by a stratified version such as the following is not satisfactory.

(SFi) $A \in I$ and there exists a clause $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ in $\mathsf{ground}(P)$ with head A such that $A_i, B_j \in I$, $l(A) \geq l(A_i)$, and $l(A) > l(B_j)$ for all i and j.

If we replace condition (WFi) by condition (SFi), then it is not guaranteed that for any given program there is a maximal model satisfying the desired properties: Consider the program consisting of the two clauses $p \leftarrow p$ and $q \leftarrow \neg p$, and the two (total) models $\{p, \neg q\}$ and $\{\neg p, q\}$, which are incomparable, and the level mapping l with $l(p) = 0$ and $l(q) = 1$.

So, in the light of Theorem 1, Definition 2 should provide a natural "stratified version" of the Fitting semantics. And indeed it does, and furthermore, the resulting semantics coincides with the well-founded semantics, which is a very satisfactory result.

Theorem 2. *Let P be a normal logic program with well-founded model M. Then M is the greatest model among all models I, for which there exists an I-partial level mapping l for P such that P satisfies (WF) with respect to I and l.*

Proof. Let M_P be the well-founded model of P and define the M_P-partial level mapping l_P as follows: $l_P(A) = \alpha$, where α is the least ordinal such that A is not undefined in $W_P\uparrow(\alpha+1)$. The proof will be established by showing the following facts: (1) P satisfies (WF) with respect to M_P and l_P. (2) If I is a model of P and l an I-partial level mapping such that P satisfies (WF) with respect to I and l, then $I \subseteq M_P$.

(1) Let $A \in \mathsf{dom}(l_P)$ and $l_P(A) = \alpha$. We consider two cases.

(Case i) If $A \in M_P$, then A satisfies (WFi) with respect to M_P and l_P, which can be shown as in the proof of Theorem 1 (1) (Case i).

(Case ii) If $\neg A \in M_P$, then $A \in U_P(W_P\uparrow\alpha)$, i.e. A is contained in the greatest unfounded set of P with respect to $W_P\uparrow\alpha$. Hence for each clause $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$, at least one of (Ui) or (Uii) holds with respect to $W_P\uparrow\alpha$ and the unfounded set $U_P(W_P\uparrow\alpha)$. If (Ui) holds, then there exists some literal $L \in \mathtt{body}$ with $\neg L \in W_P\uparrow\alpha$, hence $l_P(L) < \alpha$ and condition (WFiia) (if L is an atom), respectively condition (WFiib) (if L is a negated atom), is satisfied by A with respect to M_P and l_P. If (Uii) holds, then some (non-negated) atom B in $\mathtt{body}$ occurs in $U_P(W_P\uparrow\alpha)$. Hence $l_P(B) \leq l_P(A)$ and A satisfies (WFiia) with respect to M_P and l_P. Thus we have established that fact (1) holds.

(2) We show via transfinite induction on $\alpha = l(A)$, that whenever $A \in I$ (respectively, $\neg A \in I$), then $A \in W_P\uparrow(\alpha+1)$ (respectively, $\neg A \in W_P\uparrow(\alpha+1)$). For the base case, note that if $l(A) = 0$, then $A \in I$ implies that A occurs as the head of a fact in $\mathsf{ground}(P)$, hence $A \in W_P\uparrow 1$. If $\neg A \in I$, then consider the set U of all atoms B with $l(B) = 0$ and $\neg B \in I$. We show that U is an unfounded set of P with respect to $I = M_P$, and this suffices since it implies $\neg A \in M_P$ by $A \in U$ and the fact that M_P is a fixed point of W_P. So let $C \in U$ and let $C \leftarrow \mathtt{body}$ be a clause in $\mathsf{ground}(P)$. Since $\neg C \in I$, and $l(C) = 0$, we have that C satisfies (WFiia) with respect to I and l, so condition (Uii) is satisfied and we have that U is an unfounded set of P with respect to I. Assume now that the induction hypothesis holds for all $B \in B_P$ with $l(B) < \alpha$. We consider two cases.

(Case i) If $A \in I$, then $A \in T_P(W_P\uparrow\alpha)$, which can be shown as in the proof of Theorem 1 (2) (Case i).

(Case ii) If $\neg A \in I$, consider the set U of all atoms B with $l(B) = \alpha$ and $\neg B \in I$. We show that U is an unfounded set of P with respect to M_P, and this suffices since it implies $\neg A \in M_P$ by $A \in U$ and the fact that M_P is a fixed point of W_P. So let $C \in U$ and let $C \leftarrow \mathtt{body}$ be a clause in $\mathsf{ground}(P)$. Since $\neg C \in I$, we have that C satisfies (WFii) with respect to I and l. If there is a literal $L \in \mathtt{body}$ with $\neg L \in I$ and $l(L) < l(C)$, then by induction hypothesis we obtain $\neg L \in M_P$, so condition (Ui) is satisfied for the clause $C \leftarrow \mathtt{body}$ with respect to M_P and U. In the remaining case we have that C satisfies condition (WFiia), and there exists an atom $B \in \mathtt{body}$ with $\neg B \in I$ and $l(B) = l(C)$, hence $B \in U$,

i.e. condition (Uii) is satisfied for the clause $C \leftarrow \texttt{body}$ with respect to M_P and U. Hence U is an unfounded set of P with respect to M_P.

Example 2. The program from Example 1 has $\{a, \neg b, \neg c, \neg d, \neg e\}$ as total well-founded model. This is easily verified by choosing any level mapping l with $l(a) > l(b) = l(c) > l(d) = l(e)$ and applying Theorem 2.

As a special case of Theorem 2, we immediately obtain the following corollary, which was obtained directly in [14].

Corollary 2. *A normal logic program P has a total well-founded model if and only if there is a total model I of P and a (total) level mapping l such that P satisfies (WF) with respect to I and l.*

The characterization in Theorem 2 contrasts nicely with another characterization which can be found in [15]. There, the authors characterize the well-founded model as the *least* interpretation I such that a level-mapping property holds for all atoms which are *not false* in I.

5 Weakly Perfect Model Semantics

We have obtained new characterizations of the Fitting semantics and the well-founded semantics, and argued that the well-founded semantics is a stratified version of the Fitting semantics. Our argumentation is based on the key intuition underlying the notion of stratification, that recursion should be allowed through positive dependencies, but be forbidden through negative dependencies. As we have seen in Theorem 2, the well-founded semantics provides this for a setting in three-valued logic. Historically, a different semantics, given by the so-called weakly perfect model associated with each program, was proposed in [6] in order to carry over the intuition underlying the notion of stratification to a three-valued setting. In the following, we will characterize weakly perfect models via level mappings, in the spirit of Theorems 1 and 2. We will thus have obtained uniform characterizations of the Fitting semantics, the well-founded semantics, and the weakly perfect model semantics, which makes it possible to easily compare them.

Definition 3. *Let P be a normal logic program, I be a model of P and l be an I-partial level mapping for P. We say that P* satisfies (WS) with respect to I and l, *if each $A \in \mathsf{dom}(l)$ satisfies one of the following conditions.*

- *(WSi)* $A \in I$ *and there exists a clause $A \leftarrow L_1, \ldots, L_n$ contained in $\mathsf{ground}(P)$ such that $L_i \in I$ and $l(A) > l(L_i)$ for all i.*
- *(WSii)* $\neg A \in I$ *and for each clause $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ contained in $\mathsf{ground}(P)$ (at least) one of the following three conditions holds.*
  - *(WSiia) There exists i such that $\neg A_i \in I$ and $l(A) > l(A_i)$.*
  - *(WSiib) For all k we have $l(A) \geq l(A_k)$, for all j we have $l(A) > l(B_j)$, and there exists i with $\neg A_i \in I$.*

(WSiic) There exists j such that $B_j \in I$ and $l(A) > l(B_j)$.

We observe that the condition (WSii) in the above theorem is more general than (Fii). It is also more restrictive than (WFii), because (WFiib) and (WSiic) are the same, while (WFiia) is more general than (WSiia) and (WSiib) taken together. Informally, (WFiia) allows for recursion through positive body literals, while (WSiia) and (WSiib) allow this only if an additional requirement on the corresponding negative body literals is met.

We will see below in Theorem 3, that Definition 3 captures the weakly perfect model, in the same way in which Definitions 1 and 2 capture the Fitting model, respectively the well-founded model.

In order to proceed with this, we first need to recall the definition of weakly perfect models due to [6], and we will do this next. For ease of notation, it will be convenient to consider (countably infinite) propositional programs instead of programs over a first-order language. This is both common practice and no restriction, because the ground instantiation $\mathsf{ground}(P)$ of a given program P can be understood as a propositional program which may consist of a countably infinite number of clauses. Let us remark that our definition below differs slightly from that given in [6], and we will return to this point later. It nevertheless leads to exactly the same notion of weakly stratified program.

Let P be a (countably infinite propositional) normal logic program. An atom $A \in B_P$ *refers to* an atom $B \in B_P$ if B or $\neg B$ occurs as a body literal in a clause $A \leftarrow \mathtt{body}$ in P. *A refers negatively to B* if $\neg B$ occurs as a body literal in such a clause. We say that *A depends on B* if the pair (A, B) is in the transitive closure of the relation *refers to*, and we write this as $B \leq A$. We say that *A depends negatively on B* if there are $C, D \in B_P$ such that C refers negatively to D and one of the following holds: (1) $C \leq A$ or $C = A$ (the latter meaning identity). (2) $B \leq D$ or $B = D$. We write $B < A$ in this case. For $A, B \in B_P$, we write $A \sim B$ if either $A = B$, or A and B depend negatively on each other, i.e. if $A < B$ and $B < A$ hold. The relation $\sim$ is an equivalence relation and its equivalence classes are called *components* of P. A component is *trivial* if it consists of a single element A with $A \not< A$

Let C_1 and C_2 be two components of a program P. We write $C_1 \prec C_2$ if and only if $C_1 \neq C_2$ and for all $A_1 \in C_1$ there is $A_2 \in C_2$ with $A_1 < A_2$. A component C_1 is called *minimal* if there is no component C_2 with $C_2 \prec C_1$.

Given a normal logic program P, the *bottom stratum* $S(P)$ of P is the union of all minimal components of P. The *bottom layer* of P is the subprogram $L(P)$ of P which consists of all clauses from P with heads belonging to $S(P)$.

Example 3. In order to illustrate the definitions above, we consider the program from Example 1. The atom a refers negatively to the atom b, and depends negatively on all atoms occuring in the program. The atom d depends on e but not negatively. The atom b refers to a, c, and d and negatively to a and d. It depends negatively on itself, and hence also on all the other atoms in the program. The program has two minimal components, namely $\{d\}$ and $\{e\}$, so $\{d, e\}$ is its bottom stratum, and the bottom layer consists of the clauses $d \leftarrow e$ and $e \leftarrow d$.

Given a (partial) interpretation I of P, we define the *reduct of P with respect to I* as the program P/I obtained from P by performing the following reductions. (1) Remove from P all clauses which contain a body literal L such that $\neg L \in I$ or whose head belongs to I. (2) Remove from all remaining clauses all body literals L with $L \in I$. (3) Remove from the resulting program all non-unit clauses, whose heads appear also as unit clauses in the program.

Definition 4. *The* weakly perfect model M_P *of a program P is defined by transfinite induction as follows. Let $P_0 = P$ and $M_0 = \emptyset$. For each (countable) ordinal $\alpha > 0$ such that programs P_δ and partial interpretations M_δ have already been defined for all $\delta < \alpha$, let*

$$N_\alpha = \bigcup_{0<\delta<\alpha} M_\delta, \qquad P_\alpha = P/N_\alpha,$$

R_α *is the set of all atoms which are undefined in N_α and were eliminated from P by reducing it with respect to N_α,*

$$S_\alpha = S(P_\alpha), \quad \text{and} \quad L_\alpha = L(P_\alpha).$$

*The construction then proceeds with one of the following three cases. (1) If P_α is empty, then the construction stops and $M_P = N_\alpha \cup \neg R_\alpha$ is the (*total*)* weakly perfect model *of P. (2) If the bottom stratum S_α is empty or if the bottom layer L_α contains a negative literal, then the construction also stops and $M_P = N_\alpha \cup \neg R_\alpha$ is the (*partial*)* weakly perfect model *of P. (3) In the remaining case L_α is a definite program, and we define $M_\alpha = H \cup \neg R_\alpha$, where H is the definite (partial) model of L_α, and the construction continues.*

For every α, the set $S_\alpha \cup R_\alpha$ is called the α-th stratum *of P and the program L_α is called the* α-th layer *of P.*

A *weakly stratified program* is a program with a total weakly perfect model.

Example 4. For the program from Example 1, call it P, we obtain $N_1 = M_1 = \{\neg d, \neg e\}$ and P/N_1 consists of the clauses

$$a \leftarrow \neg b \qquad b \leftarrow c, \neg a \qquad b \leftarrow c \qquad c \leftarrow b.$$

Its least component is $\{a, b, c\}$. The corresponding bottom layer, which is all of P/N_1, contains a negative literal, so the construction stops and $M_2 = N_1 = \{\neg d, \neg e\}$ is the (partial) weakly perfect model of P.

Let us return to the remark made earlier that our definition of weakly perfect model, as given in Definition 4, differs slightly from the version introduced in [6]. In order to obtain the original definition, points (2) and (3) of Definition 4 have to be replaced as follows: (2) If the bottom stratum S_α is empty or if the bottom layer L_α *has no least two-valued model*, then the construction stops and $M_P = N_\alpha \cup \neg R_\alpha$ is the (partial) weakly perfect model of P. (3) In the remaining case L_α *has a least two-valued model*, and we define $M_\alpha = H \cup \neg R_\alpha$, where H is the partial model of L_α corresponding to its least two-valued model, and the construction continues.

The original definition is more general due to the fact that every definite program has a least two-valued model. However, while the least two-valued model of a definite program can be obtained as the least fixed point of the monotonic (and even Scott-continuous) operator T_P^+, we know of no similar result, or general operator, for obtaining the least two-valued model, if existent, of progams which are not definite. The original definition therefore seems to be rather awkward, and indeed, even in [6], when defining weakly stratified programs, the more general version was dropped in favour of requiring definite layers. So Definition 4, which will be used in the sequel, is an adaptation taking the original notion of weakly stratified program into account, and appears to be more natural.

Theorem 3. *Let P be a normal logic program with weakly perfect model M_P. Then M_P is the greatest model among all models I, for which there exists an I-partial level mapping l for P such that P satisfies (WS) with respect to I and l.*

We prepare the proof of Theorem 3 by introducing some notation, which will make the presentation much more transparent.

It will be very convenient to consider level mappings which map into *pairs* (β, n) of ordinals, where $n \leq \omega$, the least infinite ordinal. So let α be a (countable) ordinal and consider the set $\mathcal{A}$ of all pairs (β, n), where $\beta < \alpha$ and $n \leq \omega$. Of course $\mathcal{A}$ endowed with the lexicographic ordering is isomorphic to an ordinal. So any mapping from B_P to $\mathcal{A}$ can be considered to be a level mapping.

Let P be a normal logic program with (partial) weakly perfect model M_P. Then define the M_P-partial level mapping l_P as follows: $l_P(A) = (\beta, n)$, where $A \in S_\beta \cup R_\beta$ and n is least with $A \in T_{L_\beta}^+ \uparrow (n+1)$, if such an n exists, and $n = \omega$ otherwise. We observe that if $l_P(A) = l_P(B)$ then there exists α with $A, B \in S_\alpha \cup R_\alpha$, and if $A \in S_\alpha \cup R_\alpha$ and $B \in S_\beta \cup R_\beta$ with $\alpha < \beta$, then $l(A) < l(B)$.

We next define model-consistent subsumption due to [14].

Definition 5. *Let P and Q be two programs and let I be an interpretation.*

1. *If $C_1 = (A \leftarrow L_1, \ldots, L_m)$ and $C_2 = (B \leftarrow K_1, \ldots, K_n)$ are two clauses, then we say that C_1* subsumes *C_2, written $C_1 \preccurlyeq C_2$, if we have $A = B$ and $\{L_1, \ldots, L_m\} \subseteq \{K_1, \ldots, K_n\}$.*
2. *We say that P* subsumes *Q, written $P \preccurlyeq Q$, if for each clause C_1 in P there exists a clause C_2 in Q with $C_1 \preccurlyeq C_2$.*
3. *We say that P* subsumes *Q* model-consistently (with respect to I), *written $P \preccurlyeq_I Q$, if the following conditions hold. (i) For each clause $C_1 = (A \leftarrow L_1, \ldots, L_m)$ in P there exists a clause $C_2 = (B \leftarrow K_1, \ldots, K_n)$ in Q with $C_1 \preccurlyeq C_2$ and $(\{K_1, \ldots, K_n\} \setminus \{L_1, \ldots, L_m\}) \subseteq I$. (ii) For each clause $C_2 = (B \leftarrow K_1, \ldots, K_n)$ in Q with $\{K_1, \ldots, K_n\} \in I$ and $B \not\in I$ there exists a clause C_1 in P such that $C_1 \preccurlyeq C_2$.*

A clause C_1 subsumes a clause C_2 if both have the same head and the body of C_2 contains at least the body literals of C_1, e.g. $p \leftarrow q$ subsumes $p \leftarrow q, \neg r$.

A program P subsumes a program Q if every clause in P can be generated this way from a clause in Q, e.g. the program consisting of the two clauses $p \leftarrow q$ and $p \leftarrow r$ subsumes the program consisting of $p \leftarrow q, \neg s$ and $p \leftarrow r, p$. This is also an example of a model-consistent subsumption with respect to the interpretation $\{\neg s, p\}$. Concerning Example 4, we note that $P/N_1 \preccurlyeq_{N_1} P$.

Definition 5 facilitates the proof of Theorem 3 by the following lemma.

Lemma 1. *With notation from Definiton 4, we have $P/N_\alpha \preccurlyeq_{N_\alpha} P$ for all α.*

Proof. Condition 3(i) of Definition 5 holds because every clause in P/N_α is obtained from a clause in P by deleting body literals which are contained in N_α. Condition 3(ii) holds because for each clause in P with head $A \not\in N_\alpha$ whose body is true under N_α, we have that $A \leftarrow$ is a fact in P/N_α.

The next lemma establishes the induction step in part (2) of the proof of Theorem 3.

Lemma 2. *If I is a non-empty model of a (infinite propositional normal) logic program P' and l an I-partial level mapping such that P' satisfies (WS) with respect to I and l, then the following hold for $P = P'/\emptyset$.*

(a) The bottom stratum $S(P)$ of P is non-empty and consists of trivial components only.
(b) The bottom layer $L(P)$ of P is definite.
(c) The definite (partial) model N of $L(P)$ is consistent with I in the following sense: we have $I' \subseteq N$, where I' is the restriction of I to all atoms which are not undefined in N.
(d) P/N satisfies (WS) with respect to $I \setminus N$ and l/N, where l/N is the restriction of l to the atoms in $I \setminus N$.

Proof. (a) Assume there exists some component $C \subseteq S(P)$ which is not trivial. Then there must exist atoms $A, B \in C$ with $A < B$, $B < A$, and $A \not= B$. Without loss of generality, we can assume that A is chosen such that $l(A)$ is minimal. Now let A' be any atom occuring in a clause with head A. Then $A > B > A \geq A'$, hence $A > A'$, and by minimality of the component we must also have $A' > A$, and we obtain that all atoms occuring in clauses with head A must be contained in C. We consider two cases.

(Case i) If $A \in I$, then there must be a fact $A \leftarrow$ in P, since otherwise by (WSi) we had a clause $A \leftarrow L_1, \ldots, L_n$ (for some $n \geq 1$) with $L_1, \ldots, L_n \in I$ and $l(A) > l(L_i)$ for all i, contradicting the minimality of $l(A)$. Since $P = P'/\emptyset$ we obtain that $A \leftarrow$ is the only clause in P with head A, contradicting the existence of $B \not= A$ with $B < A$.

(Case ii) If $\neg A \in I$, and since A was chosen minimal with respect to l, we obtain that for each clause $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ condition (WSiib) must be satisfied with respect to I and l, and that $m = 0$. Furthermore, all A_i must be contained in C, as already noted above, and $l(A) \geq l(A_i)$ for all i by (WSiib). Also from (Case i) we obtain that no A_i can be contained in I. We have now established that for all A_i in the body of any clause with head A, we

have $l(A) = l(A_i)$ and $\neg A_i \in I$. The same argument holds for all clauses with head A_i, for all i, and the argument repeats. Now from $A > B$ we obtain that there are $D, E \in C$ with $A \geq E$ (or $A = E$), $D \geq B$ (or $D = B$), and E refers negatively to D. As we have just seen, we obtain $\neg E \in I$ and $l(E) = l(A)$. Since E refers negatively to D, there is a clause with head E and $\neg D$ contained in the body of this clause. Since (WSii) holds for this clause, there must be a literal L in the body with level less than $l(E)$, hence $l(L) < l(A)$ and $L \in C$ which is a contradiction. We thus have established that all components are trivial.

We show next that the bottom stratum is non-empty. Indeed, let A be an atom such that $l(A)$ is minimal. We will show that $\{A\}$ is a component. So assume it is not, i.e. that there is B with $B < A$. Then there exist $D_1, \ldots, D_k$, for some $k \in \mathbb{N}$, such that $D_1 = A$, D_j refers to D_{j+1} for all $j = 1, \ldots, k-1$, and D_k refers negatively to some B' with $B' \geq B$ (or $B' = B$).

We show next by induction that for all $j = 1, \ldots, k$ the following statements hold: $\neg D_j \in I$, $B < D_j$, and $l(D_j) = l(A)$. Indeed note that for $j = 1$, i.e. $D_j = A$, we have that $B < D_j = A$ and $l(D_j) = l(A)$. Assuming $A \in I$, we obtain by minimality of $l(A)$ that $A \leftarrow$ is the only clause in $P = P'/\emptyset$ with head A, contradicting the existence of $B < A$. So $\neg A \in I$, and the assertion holds for $j = 1$. Now assume the assertion holds some $j < k$. Then obviously $D_{j+1} > B$. By $\neg D_j \in I$ and $l(D_j) = l(A)$, we obtain that (WSii) must hold, and by the minimality of $l(A)$ we infer that (WSiib) must hold and that no clause with head D_j contains negated atoms. So $l(D_{j+1}) = l(D_j) = l(A)$ holds by (WSiib) and minimality of $l(A)$. Furthermore, the assumption $D_{j+1} \in I$ can be rejected by the same argument as for A above, because then $D_{j+1} \leftarrow$ would be the only clause with head D_{j+1}, by minimality of $l(D_{j+1}) = l(A)$, contradicting $B < D_{j+1}$. This concludes the inductive proof.

Summarizing, we obtain that D_k refers negatively to B', and that $\neg D_k \in I$. But then there is a clause with head D_k and $\neg B'$ in its body which satisfies (WSii), contradicting the minimality of $l(D_k) = l(A)$. This concludes the proof of statement (a).

(b) According to [6] we have that whenever all components are trivial, then the bottom layer is definite. So the assertion follows from (a).

(c) Let $A \in I'$ be an atom with $A \not\in N$, and assume without loss of generality that A is chosen such that $l(A)$ is minimal with these properties. Then there must be a clause $A \leftarrow \texttt{body}$ in P such that all literals in `body` are true with respect to I', hence with respect to N by minimality of $l(A)$. Thus `body` is true in N, and since N is a model of $L(P)$ we obtain $A \in N$, which contradicts our assumption.

Now let $A \in N$ be an atom with $A \not\in I'$, and assume without loss of generality that A is chosen such that n is minimal with $A \in T^+_{L(P)} \uparrow (n+1)$. But then there is a definite clause $A \leftarrow \texttt{body}$ in $L(P)$ such that all atoms in `body` are true with respect to $T^+_{L(P)} \uparrow n$, hence also with respect to I', and since I' is a model of $L(P)$ we obtain $A \in I'$, which contradicts our assumption.

Finally, let $\neg A \in I'$. Then we cannot have $A \in N$ since this implies $A \in I'$. So $\neg A \in N$ since N is a total model of $L(P)$.

(d) From Lemma 1, we know that $P/N \preccurlyeq_N P$. We distinguish two cases.

(Case i) If $I \setminus N \models A$, then there must exist a clause $A \leftarrow L_1, \ldots, L_k$ in P such that $L_i \in I$ and $l(A) > l(L_i)$ for all i. Since it is not possible that $A \in N$, there must also be a clause in P/N which subsumes $A \leftarrow L_1, \ldots, L_k$, and which therefore satisfies (WSi). So A satisfies (WSi).

(Case ii) If $\neg A \in I \setminus N$, then for each clause $A \leftarrow \texttt{body1}$ in P/N there must be a clause $A \leftarrow \texttt{body}$ in P which is subsumed by the former, and since $\neg A \in I$, we obtain that condition (WSii) must be satisfied by A, and by the clause $A \leftarrow \texttt{body}$. Since reduction with respect to N removes only body literals which are true in N, condition (WSii) is still met.

We can now proceed with the proof of Theorem 3.

Proof. (of Theorem 3) The proof will be established by showing the following facts: (1) P satisfies (WS) with respect to M_P and l_P. (2) If I is a model of P and l an I-partial level mapping such that P satisfies (WS) with respect to I and l, then $I \subseteq M_P$.

(1) Let $A \in \mathsf{dom}(l_P)$ and $l_P(A) = (\alpha, n)$. We consider two cases.

(Case i) If $A \in M_P$, then $A \in T^+_{L_\alpha} \uparrow (n+1)$. Hence there exists a definite clause $A \leftarrow A_1, \ldots, A_k$ in L_α with $A_1, \ldots, A_k \in T^+_{L_\alpha} \uparrow n$, so $A_1, \ldots, A_k \in M_P$ with $l_P(A) > l_P(A_i)$ for all i. Since $P/N_\alpha \preccurlyeq_{N_\alpha} P$ by Lemma 1, there must exist a clause $A \leftarrow A_1, \ldots, A_k, L_1, \ldots, L_m$ in P with literals $L_1, \ldots, L_m \in N_\alpha \subseteq M_P$, and we obtain $l_P(L_j) < l_P(A)$ for all $j = 1, \ldots, m$. So (WSi) holds in this case.

(Case ii) If $\neg A \in M_P$, then let $A \leftarrow A_1, \ldots, A_k, \neg B_1, \ldots, \neg B_m$ be a clause in P, noting that (WSii) is trivially satisfied in case no such clause exists. We consider the following two subcases.

(Subcase ii.a) Assume A is undefined in N_α and was eliminated from P by reducing it with respect to N_α, i.e. $A \in R_\alpha$. Then, in particular, there must be some $\neg A_i \in N_\alpha$ or some $B_j \in N_\alpha$, which yields $l_P(A_i) < l_P(A)$, respectively $l_P(B_j) < l_P(A)$, and hence one of (WSiia), (WSiic) holds.

(Subcase ii.b) Assume $\neg A \in H$, where H is the definite (partial) model of L_α. Since P/N_α subsumes P model-consistently with respect to N_α, we obtain that there must be some A_i with $\neg A_i \in H$, and by definition of l_P we obtain $l_P(A) = l_P(A_i) = (\alpha, \omega)$, and hence also $l_P(A_{i'}) \leq l_P(A_i)$ for all $i' \not= i$. Furthermore, since P/N_α is definite, we obtain that $\neg B_j \in N_\alpha$ for all j, hence $l_P(B_j) < l_P(A)$ for all j. So condition (WSiib) is satisfied.

(2) First note that for all models M, N of P with $M \subseteq N$ we have $(P/M)/N = P/(M \cup N) = P/N$ and $(P/N)/\emptyset = P/N$.

Let I_α denote I restricted to the atoms which are not undefined in $N_\alpha \cup R_\alpha$. It suffices to show that for all $\alpha > 0$ we have $I_\alpha \subseteq N_\alpha \cup R_\alpha$, and $I \setminus M_P = \emptyset$.

We next show by induction that if $\alpha > 0$ is an ordinal, then the following statements hold. (a) The bottom stratum of P/N_α is non-empty and consists of trivial components only. (b) The bottom layer of P/N_α is definite. (c) $I_\alpha \subseteq N_\alpha \cup R_\alpha$. (d) $P/N_{\alpha+1}$ satisfies (WS) with respect to $I \setminus N_{\alpha+1}$ and $l/N_{\alpha+1}$.

Note first that P satisfies the hypothesis of Lemma 2, hence also its consequences. So $P/N_1 = P/\emptyset$ satisfies (WS) with respect to $I \setminus N_1$ and l/N_1, and

by application of Lemma 2 we obtain that statements (a) and (b) hold. For (c), note that no atom in R_1 can be true in I, because no atom in R_1 can appear as head of a clause in P, and apply Lemma 2 (c). For (d), apply Lemma 2, noting that $P/N_2 \preccurlyeq_{N_2} P$.

For α being a limit ordinal, we can show exactly as in the proof of Lemma 2 (d), that P satisfies (WS) with respect to $I \setminus N_\alpha$ and l/N_α. So Lemma 2 is applicable and statements (a) and (b) follow. For (c), let $A \in R_\alpha$. Then every clause in P with head A contains a body literal which is false in N_α. By induction hypothesis, this implies that no clause with head A in P can have a body which is true in I. So $A \not\in I$. Together with Lemma 2 (c), this proves statement (c). For (d), apply again Lemma 2 (d), noting that $P/N_{\alpha+1} \preccurlyeq_{N_{\alpha+1}} P$.

For $\alpha = \beta + 1$ being a successor ordinal, we obtain by induction hypothesis that P/N_β satisfies the hypothesis of Lemma 2, so again statements (a) and (b) follow immediately from this lemma, and (c), (d) follow as in the case for α being a limit ordinal.

It remains to show that $I \setminus M_P = \emptyset$. Indeed by the transfinite induction argument just given we obtain that P/M_P satisfies (WS) with respect to $I \setminus M_P$ and l/M_P. If $I \setminus M_P$ is non-empty, then by Lemma 2 the bottom stratum $S(P/M_P)$ is non-empty and the bottom layer $L(P/M_P)$ is definite with definite (partial) model M. Hence by definition of the weakly perfect model M_P of P we must have that $M \subseteq M_P$ which contradicts the fact that M is the definite model of $L(P/M_P)$. Hence $I \setminus M_P$ must be empty which concludes the proof.

We obtain the following corollary, previously reported, and proven directly, in [14].

Corollary 3. *A normal logic program P is weakly stratified, i.e. has a total weakly perfect model, if and only if there is a total model I of P and a (total) level mapping l for P such that P satisfies (WS) with respect to I and l.*

We also obtain the following corollary as a trivial consequence of our uniform characterizations by level mappings.

Corollary 4. *Let P be a normal logic progam with Fitting model M_f, weakly perfect model M_p, and well-founded model M_w. Then $M_f \subseteq M_p \subseteq M_w$.*

6 Conclusions and Further Work

We have obtained new characterizations of the Fitting semantics, the well-founded semantics, and the weakly perfect model semantics, and argued that the well-founded semantics is a stratified version of the Fitting semantics. Considering that the main motivation for the introduction of (weak) stratification was to restrict recursion through negation, we notice by comparing (WFii) and (WSii) that the well-founded semantics provides a much cleaner and more convincing way of achieving this.

Our approach, using level mappings, provides a way of comparing different semantics which is an alternative to the approach taken e.g. in [4,5]. It provides

uniform characterizations, and we believe that it should be applicable to most fixed-point semantics based on monotonic operators. In particular, it should be possible to employ our methods in order to characterize different forms of well-founded semantics for (extended) disjunctive logic programs, see e.g. [16], and also to the study of fixed-point semantics of logic programming in algebraic domains, as put forward in [17,18].

Under characterizations with level mappings, as proposed in this paper, a model should be computationally tractable, in a sense which remains to be specified by further research, if the corresponding partial level mapping maps into ω, the first infinite ordinal. This is certainly the case for Datalog, and it remains to be seen whether our results can be exploited for more efficient computation of the well-founded model, as in the currently evolving paradigm of answer set programming, see e.g. [19].

References

1. Subrahmanian, V.S.: Nonmonotonic logic programming. IEEE Transactions on Knowledge and Data Engineering **11** (1999) 143–152
2. Fitting, M.: A Kripke-Kleene-semantics for general logic programs. Journal of Logic Programming **2** (1985) 295–312
3. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. Journal of the ACM **38** (1991) 620–650
4. Fitting, M.: Fixpoint semantics for logic programming — A survey. Theoretical Computer Science (200x) To appear.
5. Denecker, M., Marek, V.W., Truszynski, M.: Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. In Minker, J., ed.: Logic-based Artificial Intelligence. Kluwer Academic Publishers, Boston (2000) 127–144
6. Przymusinska, H., Przymusinski, T.C.: Weakly stratified logic programs. Fundamenta Informaticae **13** (1990) 51–65 Krzysztof R. Apt, editor, special issue of Fundamenta Informaticae on Logical Foundations of Artificial Intelligence.
7. Apt, K.R., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. In Minker, J., ed.: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, Los Altos, CA (1988) 89–148
8. Przymusinski, T.C.: On the declarative semantics of deductive databases and logic programs. In Minker, J., ed.: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, Los Altos, CA (1988) 193–216
9. Apt, K.R., Pedreschi, D.: Reasoning about termination of pure prolog programs. Information and Computation **106** (1993) 109–157
10. Fitting, M.: Metric methods: Three examples and a theorem. Journal of Logic Programming **21** (1994) 113–127
11. Seda, A.K.: Topology and the semantics of logic programs. Fundamenta Informaticae **24** (1995) 359–386
12. Hitzler, P., Seda, A.K.: Generalized metrics and uniquely determined logic programs. Theoretical Computer Science (200x) To appear.
13. Hölldobler, S., Kalinke, Y., Störr, H.P.: Approximating the semantics of logic programs by recurrent neural networks. Applied Intelligence **11** (1999) 45–58

14. Wendt, M.: Towards a unified view of the hierarchy of logic program classes. Project Thesis, Knowledge Representation and Reasoning Group, Artificial Intelligence Institute, Department of Computer Science, Dresden University of Technology (2002)
15. Lifschitz, V., McCain, N., Przymusinski, T.C., Staerk, R.F.: Loop checking and the well-founded semantics. In Marek, V.W., Nerode, A., eds.: Proceedings of the Third International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'95). Volume 928 of Lecture Notes in Artificial Intelligence., Springer, Berlin (1995) 127–142
16. Leone, N., Rullo, P., Scarello, F.: Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. Information and Computation **135** (1997) 69–112
17. Rounds, W.C., Zhang, G.Q.: Clausal logic and logic programming in algebraic domains. Information and Computation **171** (2001) 156–182
18. Hitzler, P.: Resolution and logic programming in algebraic domains: Negation and defaults. Technical Report WV-02-05, Knowledge Representation and Reasoning Group, Department of Computer Science, Dresden University of Technology, Dresden, Germany (2002)
19. Marek, V.W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In Apt, K.R., Marek, V.W., Truszczyński, M., Warren, D.S., eds.: The Logic Programming Paradigm: A 25 Year Persepective. Springer, Berlin (1999) 375–398

Axiomatization of Finite Algebras

Jochen Burghardt

GMD FIRST, Kekulestraße 7, D–12489 Berlin
jochen@first.gmd.de

Abstract. We show that the set of all formulas in n variables valid in a finite class $\mathbf{A}$ of finite algebras is always a regular tree language, and compute a finite axiom set for $\mathbf{A}$. We give a rational reconstruction of Barzdins' *liquid flow algorithm* [BB91]. We show a sufficient condition for the existence of a class $\mathbf{A}$ of *prototype algebras* for a given theory Θ. Such a set allows us to prove $\Theta \models \varphi$ simply by testing whether φ holds in $\mathbf{A}$.

1 Introduction

Abstraction is a key issue in artificial intelligence. In the setting of mathematical logic and model theory, it is concerned with the relation between *concrete* algebras and *abstract* statements about them in a formal language. For purely equational theories, the well-known construction of an initial model (e.g. [DJ90]) allows one to compute a kind of prototypical algebra for a given theory. In the other direction (i.e. from concrete algebras to theories), however, no computable procedures are yet known. While it is trivial to *check* whether a given formula is valid in a given finite algebra, it is not clear how to *find* a finite description of all valid formulas.

In 1991, Barzdin and Barzdin [BB91] proposed their *liquid-flow algorithm* which takes an incompletely given finite algebra and acquires *hypotheses* about what are probable axioms. We give a rational reconstruction of this work that is based on well-known algorithms on regular tree grammars. We give a correspondence between Barzdins' notions and grammar notions, showing that the liquid-flow algorithm in fact amounts to a combination of classical grammar algorithms (Thm. 10).

The correspondence leads to synergies in both directions: Barzdins' approach could be extended somewhat, and a classical algorithm seems to be improvable in its time complexity using the liquid-flow technique.

Next, we focus on a completely given algebra and show how to compute finite descriptions of the set of all variable-bounded formulas valid in it. This set is described by a grammar (Thm. 11) and by an axiom set (Thm. 13).

We relate our work to Birkhoff's variety theorem [MT92], which states that a class $\mathbf{A}$ of algebras can be characterized by equational axioms only up to its variety closure $\mathsf{vc}(\mathbf{A})$. If $\mathbf{A}$ is a finite class of finite algebras such that $\mathsf{vc}(\mathbf{A})$ is is finitely axiomatizable at all, we can compute an equational axiom set for it (Cor. 15).

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 222–234, 2002.

As an application in the field of automated theorem proving, we give a sufficient criterion for establishing whether a class $\mathbf{A}$ of algebras is a *prototype* class for a given theory Θ (Cor. 17). If the criterion applies, the validity of any formula φ in n variables can be decided quickly and simply by merely testing φ in $\mathbf{A}$, avoiding the search space of usual theorem proving procedures: $\Theta \models \varphi$ if and only if φ is satisfied in every $\mathcal{A} \in \mathbf{A}$.

Section 2 recalls some formal definitions. In order to make this paper self-contained, we refer well-known results on regular tree grammars that are used in the sequel. Section 3 first gives a rational reconstruction of Barzdins' liquid flow algorithm; then we show how to compute an axiom set for a finite class of finite algebras. In Sect. 4 and 5, we discuss the applications to Birkhoff characterizations and prototype algebras in theorem proving, respectively. A full version including all proofs can be found in [Bur02].

2 Definitions and Notations

Definition 1. *[Sorted term, substitution] We assume familiarity with the classic definitions of terms and substitutions in a many-sorted framework. Let $\mathcal{S}$ be a finite set of sorts. A signature Σ is a set of function symbols f, each of which has a fixed domain and range. Let $\mathcal{V}$ be an infinite set of variables, each of a fixed sort. For $S \in \mathcal{S}$ and $V \subseteq \mathcal{V}$, $\mathcal{T}_{V,S}(\Sigma)$ denotes the set of all well-sorted terms of sort S over Σ and V; let $\mathcal{T}_V(\Sigma) := \bigcup_{S \in \mathcal{S}} \mathcal{T}_{V,S}(\Sigma)$. Let $sort(t)$ denote the unique sort of a term t. $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ denotes a well-sorted substitution that maps each variable x_i to the term t_i.* □

Definition 2. *[Algebra] We consider w.l.o.g. term algebras factorized by a set of operation-defining equations. In this setting, a finite many-sorted algebra $\mathcal{A}$ of signature Σ is given by a nonempty finite set $\mathcal{A}_S$ of constants for each sort $S \in \mathcal{S}$ and a set $E_\mathcal{A}$ consisting of exactly one equation $f(a_1, \ldots, a_n) = a$ for each $f \in \Sigma$ with $f : S_1 \times \ldots \times S_n \to S$ and each $a_1 \in \mathcal{A}_{S_1}, \ldots, a_n \in \mathcal{A}_{S_n}$, where $a \in \mathcal{A}_S$. The $\mathcal{A}_S$ are just the* domains *of $\mathcal{A}$ for each sort S, while $E_\mathcal{A}$ defines the operations from Σ on these domains. Define $\Sigma_\mathcal{A} := \Sigma \cup \bigcup_{S \in \mathcal{S}} \mathcal{A}_S$. We write $(=_\mathcal{A})$ for the congruence relation induced by $E_\mathcal{A}$; each ground term $t \in \mathcal{T}_{\{\},S}(\Sigma_\mathcal{A})$ equals exactly one $a \in \mathcal{A}_S$.* □

We will only allow closed quantified equations as formulas. This is sufficient since an arbitrary formula can always be transformed into prenex normal form, and we can model predicates and junctors by functions into a dedicated sort *Bool*.

Definition 3. *[Formula, theory] For a k-tuple $\boldsymbol{x} = \langle x_1, \ldots, x_k \rangle \in \mathcal{V}^k$ such that $x_i \neq x_j$ for $i \neq j$, define $\mathcal{Q}(\boldsymbol{x}) := \{q_1 x_1 \ldots q_k x_k \mid q_1, \ldots, q_k \in \{\forall, \exists\}\}$ as the set of all quantifier prefixes over $\boldsymbol{x}$. Any expression of the form $Q : t_1 =_S t_2$ for $Q \in \mathcal{Q}(\boldsymbol{x})$ and $t_1, t_2 \in \mathcal{T}_{\boldsymbol{x},S}(\Sigma)$ is called a formula over Σ and $\boldsymbol{x}$. We will sometimes omit the index of $(=_S)$. We denote a formula by φ, and a set of formulas, also called theory, by Θ.* □

When encoding predicates and junctors using a sort *Bool*, in order to obtain an appropriate semantics[1] it is necessary and sufficient to fix the interpretation of the sort *Bool* accordingly for every algebra under consideration. Therefor, we define below the notion of an *admitted algebra*, and let the definition of sat, $\models$, etc. depend on it.

We tacitly assume that,

- when we consider only equations, each algebra is admitted, while,
- when we consider arbitrary predicates, junctors and a sort *Bool*, only algebras with an appropriate interpretation of *Bool* are admitted.

Definition 4. *[Admitted algebras] Let a signature Σ be given. Let $\mathcal{S}_{\mathrm{fix}} \subseteq \mathcal{S}$ be a set of sorts; and let Σ_{fix} be the set of all $f \in \Sigma$ that have all argument and result sorts in $\mathcal{S}_{\mathrm{fix}}$. Let a fixed Σ_{fix}-algebra $\mathcal{A}_{\mathrm{fix}}$ be given; we denote its domain sets by $\mathcal{A}_{\mathrm{fix},S}$.*

We say that a Σ-algebra $\mathcal{A}$ is admitted if $\mathcal{A}_S = \mathcal{A}_{\mathrm{fix},S}$ for each $S \in \mathcal{S}_{\mathrm{fix}}$ and $t_1 =_{\mathcal{A}} t_2 \Leftrightarrow t_1 =_{\mathcal{A}_{\mathrm{fix}}} t_2$ for all $t_1, t_2 \in \mathcal{T}_{\{\},S}(\Sigma_{\mathcal{A}})$ and $S \in \mathcal{S}_{\mathrm{fix}}$. □

Definition 5. *[Validity] For an admitted Σ-algebra $\mathcal{A}$ and a formula φ, we write $\mathcal{A}$ sat φ if φ is valid in $\mathcal{A}$, where equality symbols $(=_S)$ in φ are interpreted as identity relations on $\mathcal{A}_S$, rather than by an arbitrary congruence on it. For a class of admitted Σ-algebras $\mathbf{A}$, and a theory Θ, we similarly write $\mathbf{A}$ sat φ, $\mathcal{A}$ sat Θ, and $\mathbf{A}$ sat Θ. Define $\Theta_1 \models \Theta_2$ if $\mathcal{A}$ sat Θ_1 implies $\mathcal{A}$ sat Θ_2 for each admitted Σ-algebra $\mathcal{A}$.*

If we choose $\mathcal{S}_{\mathrm{fix}} := \{\}$, each algebra is admitted. Choosing $\mathcal{S}_{\mathrm{fix}} := \{Bool\}$, $\Sigma_{\mathrm{fix}} := \{(\neg), (\wedge), (\vee), (\rightarrow), (\leftrightarrow)\}$, and $\mathcal{A}_{\mathrm{fix}}$ as the two-element Boolean algebra, we prescribe the interpretation of Bool for each admitted algebra. □

Definition 6. *[Complete theorem sets] For a Σ-algebra $\mathcal{A}$, and a tuple $\boldsymbol{x}$ of variables as in Def. 3, define $\mathcal{TH}_{\boldsymbol{x}}(\mathcal{A}) :=$*

$$\{Q : t_1 =_S t_2 \mid Q \in \mathcal{Q}(\boldsymbol{x}),\ S \in \mathcal{S},\ t_1, t_2 \in \mathcal{T}_{\boldsymbol{x},S}(\Sigma),\ \mathcal{A}\ \mathsf{sat}\ (Q : t_1 =_S t_2)\}$$

as the set of all formulas over Σ and $\boldsymbol{x}$ that are valid in $\mathcal{A}$. The elements of $\mathcal{TH}_{\boldsymbol{x}}(\mathcal{A})$ can be considered as terms over the extended signature

$$\Sigma \cup \{(=_S) \mid S \in \mathcal{S}\} \cup \{(Q :) \mid Q \in \mathcal{Q}(\boldsymbol{x})\}.$$

For a class $\mathbf{A}$ of Σ-algebras, define $\mathcal{TH}_{\boldsymbol{x}}(\mathbf{A}) := \bigcap_{\mathcal{A} \in \mathbf{A}} \mathcal{TH}_{\boldsymbol{x}}(\mathcal{A})$. □

Example 7. The algebra $\mathcal{A}_2$, defined by $\mathcal{A}_{Nat} := \{0, 1\}$ and $E_{\mathcal{A}} = \{0+0 = 0,\ 0+1 = 1,\ 1+0 = 1,\ 1+1 = 0\}$, is a Σ-algebra for $\Sigma = \{0, (+)\}$. The set $\mathcal{TH}_{\langle x,y \rangle}(\mathcal{A}_2)$ contains the formula $\forall x \exists y : x+y=0$, but not $\forall x \exists y : x+y=1$, since $1 \not\in \Sigma$. □

[1] If in some algebra $\mathcal{A}$ we had $\mathcal{A}_{Bool} = \{a\}$ and $(true = a), (false = a) \in E_{\mathcal{A}}$, any formula φ was valid in $\mathcal{A}$.

Definition 8. *[Regular tree grammar] A regular tree grammar $\mathcal{G}$ consists of rules $N ::= f_1(N_{11}, ..., N_{1n_1}) \mid \ldots \mid f_m(N_{m1}, ..., N_{mn_m})$ or $N ::= N_1 \mid \ldots \mid N_m$ where N, N_i, N_{ij} are nonterminal symbols and $f_i \in \Sigma$. Note that n_i may be also 0. Each $f_i(N_{i1}, \ldots, N_{in_i})$ or N_i is called an alternative. N, N_i, N_{ij} are assigned a sort each that have to fit with each other and with f_i.*

The size $|\mathcal{G}|$ of $\mathcal{G}$ is its total number of alternatives. We denote the set of nonterminals of $\mathcal{G}$ by $\mathcal{N}$. The language produced by a nonterminal N of $\mathcal{G}$ is denoted by $\mathcal{L}_{\mathcal{G}}(N)$, it is a set of ground terms over Σ; if N is the start symbol of $\mathcal{G}$, we also write $\mathcal{L}(\mathcal{G})$. $\mathcal{G}$ is called deterministic if no different rules have identical alternatives.

Define the generalized height $hg(t)$ of a ground term t by

$$hg(f(t_1, \ldots, t_n)) := \max\{hg(t_1), \ldots, hg(t_n)\} + hg(f),$$

where $\max\{\} := 0$, and $hg(f) \in \mathbb{N}$ may be defined arbitrarily. For a nonterminal N of a grammar $\mathcal{G}$, define $hg(N)$ as the minimal height of any term in $\mathcal{L}_{\mathcal{G}}(N)$, it is ∞ if $\mathcal{L}_{\mathcal{G}}(N)$ is empty. □

Theorem 9. *[Properties of regular tree grammars]*

1. Incorporating *[McA92, Sect.6]*
   Given a finite many-sorted Σ-algebra $\mathcal{A}$, a grammar $\mathcal{G} = incorporate(\mathcal{A})$ of size $|E_{\mathcal{A}}|$ can be computed in time $\mathcal{O}(|E_{\mathcal{A}}|)$ such that

$$\forall S \in \mathcal{S}, a \in \mathcal{A}_S \; \exists N_a \in \mathcal{N} : \;\; \mathcal{L}_{\mathcal{G}}(N_a) = \{t \in \mathcal{T}_{\{\}}(\Sigma_{\mathcal{A}}) \mid t =_{\mathcal{A}} a\}.$$

2. Externing *[McA92, Sect.3]*
   Given a deterministic grammar $\mathcal{G}$, a set E of $|\mathcal{G}|$ ground equations can be computed in time $\mathcal{O}(|\mathcal{G}|)$ such that for all ground terms t_1, t_2:

$$t_1 =_E t_2 \;\Leftrightarrow\; \exists N \in \mathcal{N} : \; t_1, t_2 \in \mathcal{L}_{\mathcal{G}}(N),$$

   where $(=_E)$ denotes the congruence induced by E.
3. Lifting *[CDG+99, Thm.7 in Sect.1.4]*
   Given a grammar $\mathcal{G}$ and a ground substitution σ, a grammar $\mathcal{G}' = lift(\mathcal{G}, \sigma)$ of size $|\mathcal{G}| + |\mathcal{N}| \cdot |\operatorname{dom}\sigma|$ can be computed in time $\mathcal{O}(|\mathcal{G}'|)$ such that

$$\forall N \in \mathcal{N} \; \exists N' \in \mathcal{N}' : \;\; \mathcal{L}_{\mathcal{G}'}(N') = \{t \in \mathcal{T}_{\operatorname{dom}\sigma} \mid \sigma t \in \mathcal{L}_{\mathcal{G}}(N)\},$$

   where $\mathcal{N}$ and $\mathcal{N}'$ is the set of nonterminals of $\mathcal{G}$ and $\mathcal{G}'$, respectively. Note that the signature gets extended by $\operatorname{dom}\sigma$; these variables are treated as constants in $\mathcal{G}'$.
4. Intersection *[CDG+99, Sect.1.3]*
   Given n grammars $\mathcal{G}_1, \ldots, \mathcal{G}_n$, a grammar $\mathcal{G} = intersect(\mathcal{G}_1, \ldots, \mathcal{G}_n)$ of size $|\mathcal{G}_1| \cdot \ldots \cdot |\mathcal{G}_n|$ can be computed in time $\mathcal{O}(|\mathcal{G}|)$ such that for each

$$\forall N_{i_1} \in \mathcal{N}_1, \ldots, N_{i_n} \in \mathcal{N}_n \exists N_{i_1,\ldots,i_n} \in \mathcal{N} : \;\; \mathcal{L}_{\mathcal{G}}(N) = \bigcap_{j=1}^{n} \mathcal{L}_{\mathcal{G}_i}(N_{i_j}).$$

5. Restriction *[special case of 4]*
Intersection of one grammar $\mathcal{G}_1$ with a term universe $\mathcal{T}_{V,S}(\Sigma)$, such that

$$\forall N_1 \in \mathcal{N}_1 \exists N \in \mathcal{N} : \quad \mathcal{L}_{\mathcal{G}}(N) = \mathcal{L}_{\mathcal{G}_1}(N_1) \cap \mathcal{T}_{V,S}(\Sigma)$$

can be done in time $\mathcal{O}(|\,\mathcal{G}_1\,|)$ by removing all symbols not in Σ.
6. Union *[CDG+99, Sect.1.3]*
Given n grammars $\mathcal{G}_1, \ldots, \mathcal{G}_n$, a grammar $\mathcal{G} = unite(\mathcal{G}_1, \ldots, \mathcal{G}_n)$ of size $n + |\,\mathcal{G}_1\,| + \ldots + |\,\mathcal{G}_n\,|$ can be computed in time $\mathcal{O}(n)$ such that

$$\mathcal{L}(\mathcal{G}) = \bigcup_{i=1}^{n} \mathcal{L}(\mathcal{G}_i)$$

by adding one rule.
7. Composition *[Trivial]*
Given an n-ary function symbol f, a grammar $\mathcal{G}$, and nonterminals $N_1, \ldots, N_n$, a grammar $\mathcal{G}' = tag(\mathcal{G}, f(N_1, \ldots, N_n))$ of size $|\ \mathcal{G}\ | + 1$ can be computed in time $\mathcal{O}(1)$ such that

$$\mathcal{L}_{\mathcal{G}'}(N) = \{f(t_1, \ldots, t_n) \mid \bigwedge_{i=1}^{n} t_i \in \mathcal{L}_{\mathcal{G}}(N_i)\}$$

for a certain nonterminal N, by adding one rule.
8. Weight computation *[AM91, Sect.4]*
Given a grammar $\mathcal{G}$, the heights $hg(N)$ can be computed for all nonterminals $N \in \mathcal{N}$ simultaneously in time $\mathcal{O}(|\,\mathcal{N}\,|^2)$.
9. Language enumeration *[BH96, Fig.21]*
Given a grammar $\mathcal{G}$ and the heights of all nonterminals, the elements of $\mathcal{L}_{\mathcal{G}}(N)$ can be enumerated in order of increasing height in time linear in the sum of their sizes by a simple PROLOG *program.* □

3 Equational Theories of Finite Algebras

First, we give a rational reconstruction of the *liquid flow algorithm* of Barzdin and Barzdin [BB91]. They use labeled graphs to compute an axiom set from an incompletely given finite algebra. Their approach can be reformulated in terms of regular tree languages using the correspondence of notions shown in Fig. 1. Our following theorem corresponds to their main result, Thm. 2. It is in fact a slight extension, as it allows for sorts and for substitutions that map several variables to the same value.

On the other hand, the *liquid flow algorithm* turns out to be an improvement of the weight computation algorithm from Thm. 9.8. Both are fixpoint algorithms, and identical except for minor, but important, modifications. The algorithm from Thm. 9.8 has a complexity of $\mathcal{O}(|\,\mathcal{N}\,|^2)$, while Barzdins' algorithm runs in $\mathcal{O}(|\,\mathcal{N}\,|)$, exploiting the fact that always $hg(f) \leqslant 1$ and therefor the first value $< \infty$ assigned to some $hg(N)$ must be its final one already. Since

in each cycle at least one N must change its assigned hg value[2], by an appropriate incremental technique *(water front)*, linear complexity can be achieved. A formal complexity proof of this improved grammar fixpoint algorithm, extended to somewhat more general weight definitions, shall appear in [Bur02].

Theorem 10. *[Reconstruction of Barzdin] Given a Σ-algebra $\mathcal{A}$, domain elements $b_1, ..., b_n \in \mathcal{A}_{S_0}, a_{11}, ..., a_{n1} \in \mathcal{A}_{S_1}, \ldots, a_{1k}, ..., a_{nk} \in \mathcal{A}_{S_k}$, and defining $\sigma_i = \{x_1 \mapsto \not{a}_{i1}, \ldots, x_k \mapsto \not{a}_{ik}\}$ for $i = 1, \ldots, n$ and $V := \{x_1, \ldots, x_k\}$, the set of terms $T = \{t \in \mathcal{T}_{V,S_0}(\Sigma) \mid \bigwedge_{i=1}^{n} \sigma_i t =_{\mathcal{A}} b_i\}$ is a regular tree language. A grammar for it can be computed in time $\mathcal{O}(\mid E_{\mathcal{A}} \mid^n)$. After computing nonterminal weights in time $\mathcal{O}(\mid E_{\mathcal{A}} \mid^{2n})$, the language elements can be enumerated in order of increasing height in linear time.*

Proof. Using the notions of Thm. 9, let $\mathcal{G}_0 := incorporate(\mathcal{A})$, and $\mathcal{G}_i := lift(\mathcal{G}_0, \sigma_i)$ for $i = 1, \ldots, n$. We have

$$\mathcal{L}_{\mathcal{G}_i}(N_a) = \{t \in \mathcal{T}_V(\Sigma_A) \mid \sigma_i t =_{\mathcal{A}} a\}.$$

Let $\mathcal{G} := intersect(\mathcal{G}_1, \ldots, \mathcal{G}_n, \mathcal{T}_V(\Sigma))$, then

$$\mathcal{L}_{\mathcal{G}}(N_{a_1,\ldots,a_n}) = \{t \in \mathcal{T}_V(\Sigma) \mid \bigwedge_{i=1}^{n} \sigma_i t =_{\mathcal{A}} a_i\}.$$

Hence $T = \mathcal{L}_{\mathcal{G}}(N_{b_1,\ldots,b_n})$. Note that $\mathcal{G}$ itself does not depend on $b_1, \ldots, b_n$. Compute the height of all nonterminals of $\mathcal{G}$ using Thm. 9.8, using $hg(x_i) := 0$ for $x_i \in V$ and $hg(f) := 1$ for $f \in \Sigma$. Use Thm. 9.9 to enumerate the terms of $\mathcal{L}_{\mathcal{G}}(N_{b_1,\ldots,b_n})$. □

Barzdin and Barzdin allow to specify an algebra incompletely, since their main goal is to acquire *hypotheses* about what are probable axioms.

We now investigate the special case that the substitutions $\sigma_1, \ldots, \sigma_n$ in Thm. 10 describe *all* possible assignments of algebra domain elements to the variables $x_1, \ldots, x_k$. This way, we obtain *certainty* about the computed axioms – they are guaranteed to be valid in the given algebra.

Theorem 11. *[Computing complete theorem sets] Let $\boldsymbol{x} = \langle x_1, \ldots, x_k \rangle$ be a k-tuple of variables, $\mathcal{A}$ be a finite Σ-algebra and $\mathbf{A}$ a finite class of finite Σ-algebras, then $\mathcal{TH}_{\boldsymbol{x}}(\mathcal{A})$ and $\mathcal{TH}_{\boldsymbol{x}}(\mathbf{A})$ are regular tree languages.*

Proof (sketch). Define $\mathcal{A}_{\boldsymbol{x}} := \mathcal{A}_{sort(x_1)} \times \ldots \times \mathcal{A}_{sort(x_k)}$. For each $\boldsymbol{a} \in \mathcal{A}_{\boldsymbol{x}}$, let $\sigma_{\boldsymbol{a}} := \{x_1 \mapsto \not{a}_1, \ldots, x_k \mapsto \not{a}_k\}$. Let $\mathcal{G} = incorporate(\mathcal{A})$ and $\mathcal{G}_{\boldsymbol{a}} = intersect(lift(\mathcal{G}, \sigma_{\boldsymbol{a}}), \mathcal{T}_{\boldsymbol{x}}(\Sigma))$ for $\boldsymbol{a} \in \mathcal{A}_{\boldsymbol{x}}$. For $S \in \mathcal{S}$ and $a \in \mathcal{A}_S$, let $\mathcal{G}_{\boldsymbol{a},a} = tag(\mathcal{G}_{\boldsymbol{a}}, N_a =_S N_a)$, where $(=_S)$ is a new binary infix function symbol. Let $\mathcal{G}'_{\boldsymbol{a}} = unite(\{\mathcal{G}_{\boldsymbol{a},a} \mid S \in \mathcal{S}, a \in \mathcal{A}_S\})$ for each $\boldsymbol{a} \in \mathcal{A}_{\boldsymbol{x}}$. We have

$$\mathcal{L}(\mathcal{G}'_{\boldsymbol{a}}) = \{t_1 =_S t_2 \mid S \in \mathcal{S},\ t_1, t_2 \in \mathcal{T}_{\boldsymbol{x},S}(\Sigma),\ \sigma_{\boldsymbol{a}} t_1 =_{\mathcal{A}} \sigma_{\boldsymbol{a}} t_2\}.$$

[2] Unless the fixpoint has been reached already

Barzdin [BB91]	**Tree Grammars**
sample P	equations $E_{\mathcal{A}}$ from Def. 2
open term, level	term in $\mathcal{T}_{\mathcal{V}}(\Sigma)$, height
closed term	term in $\mathcal{T}_{\{\}}(\Sigma_{\mathcal{A}})$
sample graph	grammar $\mathcal{G} = incorporate(\mathcal{A})$
domain node	nonterminal N_a
functional node	expression $f(N_{a_1}, \ldots, N_{a_n})$
upper node	N_a, if $N_a ::= \ldots f(N_{a_1}, \ldots, N_{a_n}) \ldots$
lower nodes	$N_{a_1}, \ldots, N_{a_n}$
node weight	language height $hg(N)$ from Def. 8
chain of dotted arcs	rule rhs with alternatives ordered by increasing height
annotated sample graph	grammar with heights of nonterminals obtained from Thm. 9.8
liquid-flow algorithm	(improved) height computation algorithm from Thm. 9.8
α-term	term in $\mathcal{T}_{\{\}}(\Sigma_{\mathcal{A}}) \cap \bigcup_{a \in \mathcal{A}_S} \mathcal{L}(N_a)$
minimal α-term of domain node d	term $t \in \mathcal{L}(N_d)$ of minimal height
minimal α-term of functional node	term $t \in \mathcal{L}(f(N_{a_1}, \ldots, N_{a_n}))$ of minimal height
Theorem 2	Theorem 10
Theorem 1	Theorem 10 for $n = 1$

Fig. 1. Correspondence of notions between [BB91] and regular tree grammars

Now, for each $Q \in \mathcal{Q}(\boldsymbol{x})$, apply set operations corresponding to Q to the $\mathcal{G}_{\boldsymbol{a}}$; e.g. if $k = 2$ and $Q = (\forall x_1 \exists x_2)$, let

$$\mathcal{G}_Q = intersect(\{unite(\{\mathcal{G}_{a_1,a_2} \mid a_2 \in \mathcal{A}_{sort(a_2)}\}) \mid a_1 \in \mathcal{A}_{sort(a_1)}\}).$$

In general, we get

$$\mathcal{L}(\mathcal{G}_Q) = \{t_1 =_S t_2 \mid S \in \mathcal{S},\ t_1, t_2 \in \mathcal{T}_{\boldsymbol{x},S}(\Sigma),\ \mathcal{A} \ \mathsf{sat}\ (Q : t_1 = t_2)\}.$$

For each $Q \in \mathcal{Q}(\boldsymbol{x})$, let $\mathcal{G}'_Q = tag(\mathcal{G}_Q, Q : \mathcal{G}_Q)$, where $(Q:)$ is a new unary prefix function symbol. Let $\mathcal{G}' = unite(\{\mathcal{G}'_Q \mid Q \in \mathcal{Q}(\boldsymbol{x})\})$, then $\mathcal{TH}_{\boldsymbol{x}}(\mathcal{A}) = \mathcal{L}(\mathcal{G}')$. From this, we immediately get a grammar for $\mathcal{TH}_{\boldsymbol{x}}(\mathbf{A})$ by Thm. 9.4. □

Example 12. Let us compute $\mathcal{TH}_{\boldsymbol{x}}(\mathcal{A}_2)$ for the algebra $\mathcal{A}_2$ from Exm. 7 and $\boldsymbol{x} = \langle x, y \rangle$. We have the substitutions $\sigma_{00}, \sigma_{01}, \sigma_{10}, \sigma_{11}$ and use the naming convention

$$\mathcal{L}(N_{ijkl}) = \mathcal{L}(\mathcal{G}_{00,i}) \cap \mathcal{L}(\mathcal{G}_{01,j}) \cap \mathcal{L}(\mathcal{G}_{10,k}) \cap \mathcal{L}(\mathcal{G}_{11,l}),$$

e.g. $\mathcal{L}(N_{0011}) = \{t \in \mathcal{T}_{\boldsymbol{x}}(\Sigma) \mid \mathcal{A}_2 \ \mathsf{sat}\ (\sigma_{00}t = \sigma_{01}t = 0 \wedge \sigma_{10}t = \sigma_{11}t = 1)\}$. A "*" may serve as *don't care symbol*, e.g. $\mathcal{L}(N_{i*k*}) = \mathcal{L}(\mathcal{G}_{00,i}) \cap \mathcal{L}(\mathcal{G}_{10,k})$.

From Thm. 11, after incorporating, lifting, and restricting, we obtain e.g. $\mathcal{G}_{00}$, with nonterminals N_{0***} and N_{1***}. As $\mathcal{L}(N_{1***})$ turns out to be empty, we simply have

$$N_{0***} ::= 0 \mid x \mid y \mid N_{0***} + N_{0***}.$$

$$
\begin{array}{lll}
N_{0000} ::= 0 \mid N_{0000} + N_{0000} & \mid N_{0011} + N_{0011} & \mid N_{0101} + N_{0101} \mid N_{0110} + N_{0110} \\
N_{0011} ::= x \mid N_{0000} + N_{0011} & \mid N_{0011} + N_{0000} & \mid N_{0101} + N_{0110} \mid N_{0110} + N_{0101} \\
N_{0101} ::= y \mid N_{0000} + N_{0101} & \mid N_{0011} + N_{0110} & \mid N_{0101} + N_{0000} \mid N_{0110} + N_{0011} \\
N_{0110} ::= N_{0000} + N_{0110} & \mid N_{0011} + N_{0101} & \mid N_{0101} + N_{0011} \mid N_{0110} + N_{0000} \\
N_{0*0*} ::= N_{0000} & \mid N_{0001} & \mid N_{0100} \mid N_{0101} \\
N_{0*1*} ::= \ldots & & \\
N_{\forall\forall} ::= N_{0000} = N_{0000} & \mid N_{0011} = N_{0011} & \mid N_{0101} = N_{0101} \mid N_{0110} = N_{0110} \\
N_{\forall\exists} ::= N_{0*0*} = N_{0*0*} & \mid N_{0*1*} = N_{0*1*} & \mid N_{0**0} = N_{0**0} \mid N_{0**1} = N_{0**1} \\
\quad \mid N_{*00*} = N_{*00*} & \mid N_{*01*} = N_{*01*} & \mid N_{*10*} = N_{*10*} \mid N_{*11*} = N_{*11*} \\
\quad \mid N_{*0*0} = N_{*0*0} & \mid N_{*0*1} = N_{*0*1} & \mid N_{*1*0} = N_{*1*0} \mid N_{*1*1} = N_{*1*1} \\
N_{\exists\forall} ::= N_{00**} = N_{00**} & \mid N_{01**} = N_{01**} & \\
\quad \mid N_{**00} = N_{**00} & \mid N_{**01} = N_{**01} & \mid N_{**10} = N_{**10} \mid N_{**11} = N_{**11} \\
N_{\exists\exists} ::= N_{0***} = N_{0***} & \mid N_{*0**} = N_{*0**} & \mid N_{*1**} = N_{*1**} \\
\quad \mid N_{**0*} = N_{**0*} & \mid N_{**1*} = N_{**1*} & \mid N_{***0} = N_{***0} \mid N_{***1} = N_{***1} \\
N ::= \forall x \forall y : N_{\forall\forall} & \mid \forall x \exists y : N_{\forall\exists} & \mid \exists x \forall y : N_{\exists\forall} \mid \exists x \exists y : N_{\exists\exists}
\end{array}
$$

Fig. 2. Grammar $\mathcal{G}'$ for $\mathcal{TH}_{\boldsymbol{x}}(\mathcal{A})$ in Exm. 12

$$
\begin{array}{l}
\hline
\forall x \forall y : \quad N \\
\hline
\forall x \forall y : \quad N_{\forall\forall} \\
\hline
\forall x \forall y : \quad N_{0110} \quad = \quad N_{0110} \\
\hline
\forall x \forall y : \quad \underline{N_{0011}} + \underline{N_{0101}} \quad = \quad \underline{N_{0101}} + \underline{N_{0011}} \\
\forall x \forall y : \quad x + y = y + x
\end{array}
\qquad
\begin{array}{l}
\hline
\forall x \exists y : \quad N \\
\hline
\forall x \exists y : \quad N_{\forall\exists} \\
\hline
\forall x \exists y : \quad \underline{N_{0**0}} \quad = \quad \underline{N_{0**0}} \\
\forall x \exists y : \quad \underline{N_{0110}} \quad = \quad \underline{N_{0000}} \\
\forall x \exists y : \quad \underline{N_{0011}} + \underline{N_{0101}} = 0 \\
\forall x \exists y : \quad x + y = 0
\end{array}
$$

Fig. 3. Example derivations from $\mathcal{G}'$ in Exm. 12

We obtain $\mathcal{G}'_{00}$ by just adding the rule $N_{=} ::= (N_{0***} = N_{0***})$. To compute $\mathcal{G}_{\forall\forall}$, we build all intersections N_{ijkl} without "*"; only four of them turn out to be nonempty, their rules are shown in Fig. 2. The grammar $\mathcal{G}_{\forall\forall}$ consists of these rules and an additional one for its start symbol $N_{\forall\forall}$. The grammars $\mathcal{G}_{\forall\exists}$, $\mathcal{G}_{\exists\forall}$, and $\mathcal{G}_{\exists\exists}$ have similar starting rules, which use only nonterminals N_{ijkl} containing a "*". Since the rules for the latter are trivial, only the one for N_{0*0*} is shown. Finally, the grammar $\mathcal{G}'$ for $\mathcal{TH}_{\boldsymbol{x}}(\mathcal{A}_2)$ consists of all these rules and an additional one for its start symbol N. Figure 3 show some example derivations. □

Theorem 13. *[Computing complete axiom sets] The sets $\mathcal{TH}_{\boldsymbol{x}}(\mathcal{A})$ and $\mathcal{TH}_{\boldsymbol{x}}(\mathbf{A})$ obtained from Thm. 11 can be represented as the deductive closure of a finite set of formulas, called $\mathcal{AX}_{\boldsymbol{x}}(\mathcal{A})$ and $\mathcal{AX}_{\boldsymbol{x}}(\mathbf{A})$, respectively. We have:*

$$
\begin{array}{l}
\forall t_1, t_2 \in \mathcal{T}_{\boldsymbol{x}}(\Sigma) : \ \mathcal{A} \ \mathsf{sat} \ t_1 = t_2 \Leftrightarrow \mathcal{AX}_{\boldsymbol{x}}(\mathcal{A}) \models t_1 = t_2 \ , \ \textit{and} \\
\forall t_1, t_2 \in \mathcal{T}_{\boldsymbol{x}}(\Sigma) : \mathbf{A} \ \mathsf{sat} \ t_1 = t_2) \Leftrightarrow \mathcal{AX}_{\boldsymbol{x}}(\mathbf{A}) \models t_1 = t_2 \ .
\end{array}
$$

Proof (sketch). First, we consider the purely universal formulas in $\mathcal{TH}_{\boldsymbol{x}}(\mathcal{A})$. Using the notions of Thm. 11, the grammar $\mathcal{G}_{\forall\ldots\forall}$ is deterministic since no union operations were involved in its construction. Using Thm. 9.2, we get a finite set

$E_{\forall\dots\forall}$ of equations, each of which we compose with the appropriate universal quantifier prefix ($\forall x_1 \dots \forall x_k$:). The resulting formula set $E'_{\forall\dots\forall}$ implies any purely universal equation valid in $\mathcal{A}$. By construction of $E_{\forall\dots\forall}$, it can reduce each term t in any quantified equation in $\mathcal{TH}_{\boldsymbol{x}}(\mathcal{A})$ to a unique normal form. Let NF denote the set of all those normal forms; it is finite since $|\mathcal{N}|$ is finite.

Next, for any quantifier prefix Q containing some "$\exists$", let

$$E'_Q := \{Q : t_{1n} = t_{2n} \mid t_{1n}, t_{2n} \in NF,\ (t_{1n} = t_{2n}) \in \mathcal{L}(\mathcal{G}_Q),\ t_{1n} \not= t_{2n}\}.$$

Any formula $Q : t_1 = t_2$ in $\mathcal{L}(\mathcal{G}'_Q)$ can then be deduced from $\forall\dots\forall : t_1 = t_{1n}$ and $\forall\dots\forall : t_2 = t_{2n}$ in $E'_{\forall\dots\forall}$ and $Q : t_{1n} = t_{2n}$ in E'_Q, where t_{1n} and t_{2n} are the normal forms of t_1 and t_2, respectively.

Finally, let $\mathcal{AX}_{\boldsymbol{x}}(\mathcal{A}) = \bigcup_{Q\in\mathcal{Q}(\boldsymbol{x})} E'_Q$. The proof for $\mathcal{AX}_{\boldsymbol{x}}(\mathbf{A})$ is similar. □

Observe that the variables in $\boldsymbol{x}$ are introduced as constants into the grammars, hence $E_{\forall\dots\forall}$ in the above proof is a set of ground equations. A closer look at the algorithm referred by Thm. 9.2 reveals that it generates in fact a Noetherian ground-rewriting system assigning unique normal forms. Anyway, no proper instance of any formula from $\mathcal{AX}_{\boldsymbol{x}}(\mathcal{A})$ is needed to derive any one in $\mathcal{TH}_{\boldsymbol{x}}(\mathcal{A})$. By permitting proper instantiations, we may delete formulas that are instances of others, thus reducing their total number significantly. To find such subsumed formulas, an appropriate indexing technique may be used, see e.g. [Gra94].

$$\begin{array}{lllllllll}
\forall\forall: & N_{0000}: & \boxed{0} & = & 0+0 & = & \boxed{x+x} & = & y+y & = (x+y)+(x+y) \\
& N_{0011}: & \boxed{x} & = & 0+x & = & \boxed{x+0} & = y+(x+y) & = & \boxed{(x+y)+y} \\
& N_{0101}: & y & = & 0+y & = x+(x+y) & = & y+0 & = & (x+y)+x \\
& N_{0110}: & \boxed{x+y} & = 0+(x+y) & & & = & \boxed{y+x} & = & (x+y)+0
\end{array}$$

$$\begin{array}{llllll}
\forall\exists: & N_{0*0*}: 0=y & N_{0**0}: \boxed{0=x+y} & N_{0*1*}: x=x+y & N_{0**1}: x=y & \\
\exists\forall: & N_{00**}: 0=x & N_{01**}: y=x+y & & & \\
\exists\exists: & N_{0***}: 0=x=y=x+y & & N_{*0**}: 0=x & N_{*1**}: y=x+y & \\
& N_{**0*}: 0=y & N_{**1*}: x=x+y & N_{***0}: 0=x+y & N_{***1}: x=y &
\end{array}$$

Fig. 4. Axioms $\mathcal{AX}_{\boldsymbol{x}}(\mathcal{A})$ in Exm. 14

Example 14. Continuing Exm. 12, and referring to the notions the proof of of Thm. 13, we obtain the set $E_{\forall\forall}$ shown at the top of Fig. 4, where we chose the normal form of N_{0000}, N_{0011}, N_{0101}, and N_{0110} as 0, x, y, and $x+y$, respectively[3], which are each of minimal size. From each rule alternative in Fig. 2, we get one equation that is universally valid in $\mathcal{A}$. The equations between marked terms remain nontrivial if instantiations are allowed.

[3] The algorithm of Thm. 9.2 can easily be modified to work with arbitrary chosen normal forms instead of external constants.

For each of the N_{ijkl} with a "*", we check which of the above 4 normal forms are member of $\mathcal{L}(N_{ijkl})$. This can be decided quickly by matching the index; e.g. N_{0*0*} contains 0 and y since $0*0*$ matches with 0000 and 0101. Each pair of normal forms in the same $\mathcal{L}(N_{ijkl})$ gives rise to an equation, shown at the bottom of Fig. 4. Note that equations between terms of different N_{ijkl} are neither in $\mathcal{TH}_{\boldsymbol{x}}(\mathcal{A})$ nor in $\mathcal{AX}_{\boldsymbol{x}}(\mathcal{A})$. For example, "crossing" N_{0*0*} and N_{0**0} yields the forbidden formula $\forall x \exists y : y = x + y$ which does not hold in $\mathcal{A}$.

After removing all redundancies[4], we get

$$\{ \quad \forall x : 0 = x+x, \quad \forall x \forall y : x+y = y+x, \quad \forall x \exists y : 0 = x+y, \\ \forall x : x = x+0, \quad \forall x \forall y : x = (x+y)+y \qquad\qquad \}$$

as a set of formulas implying every closed formula over $\{x, y, 0, (+), (=)\}$ that is valid in $\mathcal{A}$. Note that the associativity law is not implied, since it requires 3 variables. □

4 Application to Equational Theories

We now show some consequences of axiomatization properties in a purely equational setting. Remember our convention made before Def. 4, that in this setting, each algebra is considered to be admitted. We restrict $\mathcal{TH}$ and $\mathcal{AX}$ to the set

$$\mathcal{U} := \{\forall \ldots \forall : t_1 =_S t_2 \mid S \in \mathcal{S},\ t_1, t_2 \in \mathcal{T}_{\mathcal{V},S}(\Sigma)\},$$

which is trivially a regular tree language.

For a given signature Σ and a class $\mathbf{A}$ of Σ-algebras, let $\mathsf{vc}(\mathbf{A})$ denote the smallest variety containing $\mathbf{A}$, i.e., the class of all Σ-algebras obtainable from algebras in $\mathbf{A}$ by building subalgebras, Cartesian products, and homomorphic images. For a set E of equations, let $\mathsf{Mod}(E)$ denote the class of all Σ-algebras $\mathcal{A}$ with $\mathcal{A}$ sat E. From Birkhoff's variety theorem [MT92], it is well known that each class $\mathbf{A}$ of algebras can be characterized by universal equations only up to its variety closure $\mathsf{vc}(\mathbf{A})$. However, it is not clear in general how to find such an axiom set E with $\mathsf{Mod}(E) = \mathsf{vc}(\mathbf{A})$.

If $\mathbf{A}$ is a finite class of finite algebras, we can at least construct an increasing sequence of tree languages characterizing $\mathsf{vc}(\mathbf{A})$ *in the limit* (Cor. 15). Whenever there exists any finite axiom set E for $\mathsf{vc}(\mathbf{A})$ at all, the sequence of corresponding model classes eventually becomes stationary, and, using Thm. 13, we can obtain a finite axiom set that uniquely characterizes $\mathsf{vc}(\mathbf{A})$. However, convenient criteria for detecting if and when the sequence becomes stationary are still unknown.

Corollary 15. *[Variety Characterization] For any finite class of finite algebras* $\mathbf{A}$*, we can compute a sequence* $TH_1 \subseteq TH_2 \subseteq \ldots$ *of sets of universal equations such that* $\mathsf{vc}(\mathbf{A}) = \mathsf{Mod}(\bigcup_{i=1}^{\infty} TH_i)$*. If* $\mathsf{vc}(\mathbf{A}) = \mathsf{Mod}(E)$ *for any finite* E*, we already have* $\mathsf{Mod}(TH_n) = \mathsf{vc}(\mathbf{A})$ *for some* $n \in \mathbb{N}$*. In this case, we can compute a finite axiom set for* $\mathsf{vc}(\mathbf{A})$ *from* TH_n*.* □

[4] E.g. $\exists x \forall y : y = x + y$ follows from $\forall x : x = x + 0$ and $\forall x \forall y : x + y = y + x$.

Proof. Assuming $\mathcal{V} = \{x_1, x_2, \ldots\}$, let $\boldsymbol{x}_i := \langle x_1, \ldots, x_n \rangle$ for $i \in I\!N$. Let $TH_i := \mathcal{TH}_{\boldsymbol{x}_i}(\mathbf{A}) \cap \mathcal{U}$ and $TH_\infty := \bigcup_{i=1}^{\infty} TH_i$; then, $TH_i \subseteq TH_{i+1} \subseteq TH_\infty$. By Thm. 11, TH_∞ consists of all universal equations that hold in $\mathbf{A}$.

By Birkhoff's variety theorem, $\mathsf{vc}(\mathbf{A}) = \mathsf{Mod}(E)$ for some set E of equations. Since $\mathbf{A}$ sat E, we have $E \subseteq TH_\infty$, hence $\mathsf{Mod}(TH_\infty) \subseteq \mathsf{Mod}(E) = \mathsf{vc}(\mathbf{A})$. Vice versa, we have $\mathsf{vc}(\mathbf{A}) \subseteq \mathsf{Mod}(TH_\infty)$, since $\mathbf{A} \subseteq \mathsf{Mod}(TH_\infty)$, and $\mathsf{Mod}(TH_\infty)$ is closed wrt. subalgebras, products, and homomorphic images.

If E is finite, let $n \in I\!N$ such that all variables in E occur in $\boldsymbol{x}_n$, then $\mathsf{Mod}(TH_n) = \mathsf{vc}(\mathbf{A})$ as above. □

5 Application to Theorem Proving

Next, we extend our results to arbitrary formulas of first-order predicate logic. This can easily be achieved by including a sort *Bool* and encoding predicates and junctors as functions to *Bool*. We admit only algebras with an appropriate interpretation of *Bool*, cf. the convention before Def. 4.

Thus, the equation set $\mathcal{TH}_{\boldsymbol{x}}(\mathcal{A})$, and $\mathcal{AX}_{\boldsymbol{x}}(\mathcal{A})$ corresponds to the set of all formulas in $\boldsymbol{x}$ valid in $\mathcal{A}$, and a finite axiomatization of it, respectively. Moreover, we can arbitrarily restrict the set of junctors that may occur in a formula. Note, however, that we cannot get rid of any equality predicate[5], as they are core components of our approach, cf. Def. 6. Hence, we cannot compute the set of all Horn formulas valid in a given algebra.

$$
\begin{array}{r}
x+x = 0 \\
0+x = x \\
x+0 = x \\
(y+x)+y = x \\
(x+y)+y = x \\
(x+x)+y = y \\
(x+y)+x = y \\
y+x = x+y
\end{array}
\qquad
\begin{array}{r}
0<x \wedge 0<y \wedge 0<x+y \leftrightarrow \mathit{false} \\
0<x \wedge 0<y \wedge x<y \leftrightarrow \mathit{false} \\
0<x \wedge 0<y \wedge y<x \leftrightarrow \mathit{false} \\
0<x \wedge x<y \leftrightarrow \mathit{false} \\
x<y \wedge y<x \leftrightarrow \mathit{false} \\
x<0 \leftrightarrow \mathit{false} \\
x<x \leftrightarrow \mathit{false} \\
x+y<x \leftrightarrow 0<x \wedge 0<y \\
x+y<y \leftrightarrow 0<x \wedge 0<y \\
0<x \wedge 0<x+y \leftrightarrow y<x \\
y<x \wedge (0<x \vee x<y) \leftrightarrow y<x \\
y<x+y \leftrightarrow y<x
\end{array}
\qquad
\begin{array}{r}
(0<x \wedge 0<y) \vee y<x \leftrightarrow 0<x \\
0<y \wedge 0<x+y \leftrightarrow x<y \\
x<x+y \leftrightarrow x<y \\
0<y \wedge (0<x \vee x<y) \leftrightarrow 0<y \\
(0<x \wedge 0<y) \vee x<y \leftrightarrow 0<y \\
0<x+y \wedge (0<x \vee x<y) \leftrightarrow 0<x+y \\
x<y \vee y<x \leftrightarrow 0<x+y \\
(0<x \wedge 0<y) \vee 0<x+y \leftrightarrow 0<x \vee x<y \\
0<x \vee 0<y \leftrightarrow 0<x \vee x<y \\
0<x \vee 0<x+y \leftrightarrow 0<x \vee x<y \\
0<y \vee 0<x+y \leftrightarrow 0<x \vee x<y \\
0<y \vee y<x \leftrightarrow 0<x \vee x<y
\end{array}
$$

Fig. 5. Axiom Set in Exm. 16

Example 16. As an example of computed predicate-logic axiomatizations, consider ($I\!N$ *mod* 2) with one function ($+$) and one predicate ($<$), where the sort *Bool* is interpreted by the two-element Boolean algebra, as required.

Any universally quantified formula in the variables x, y and with $\wedge$ and $\vee$ as the only logical junctors that holds in ($I\!N$ *mod* 2) follows from the set of formulas given in Fig. 5; formulas with other quantifier prefixes are omitted.

Pure ground formulas and formulas that are instances of others have been manually deleted, as well as formulas that follow from propositional tautologies

[5] Logical equivalence in *Bool*

or symmetry of equality. Equations in Nat are listed first, with index $_{Nat}$ omitted, followed by equations in $Bool$, $=_{Bool}$ written as $\leftrightarrow$. Note that $(=_{Nat})$ is *not* contained in the signature of this example but was introduced by our method; consequently, no equality predicate appears in the equations of sort $Bool$, e.g. in a law like $x < y \vee x = y \vee y < x \leftrightarrow true$.

No formula at all can be reduced to $true$ by the axioms in Fig. 5, indicating that no valid statements about $(\mathbb{N}\ mod\ 2)$ can be expressed in Σ, except for trivial propositional instances like $true \vee x < y$. In fact, every expressible formula (i.e. term in $\mathcal{T}_{\{x,y\},Bool}(\Sigma)$) can be falsified by instantiating both x and y to 0. □

In Cor. 17, we give an application in the field of automated theorem proving. Here, it is common practice to test a conjecture φ in a finite class $\mathbf{A}$ of models of the background theory Θ before attempting to prove φ from Θ. If the test fails, it is clear that $\Theta \models \varphi$ cannot hold. If the test succeeds, i.e. $\mathbf{A}$ sat φ, we are usually still faced with the task of proving $\Theta \models \varphi$.

We call $\mathbf{A}$ a class of prototype algebras for Θ, if from a succeeding test we always can conclude the validity of $\Theta \models \varphi$. In this case, we can decide quickly whether Θ entails a formula φ, merely by testing whether φ holds in each member of $\mathbf{A}$.

Corollary 17 provides a sufficient criterion for establishing the existence of prototype algebras for an equational – or by the above argument – first-order predicate-logic theory Θ. Since we cannot deal with arbitrarily many variables, we have to restrict the syntactic class of φ to $\mathcal{T}_{\boldsymbol{x}}(\Sigma \cup \{(=_{Bool})\})$ for some finite tuple $\boldsymbol{x}$ of variables.

Example 18 gives a set of prototype algebras for an equational theory, and at the same time shows that they do not exist for arbitrary theories. It remains to be seen whether it is feasible to extend the prototype approach by adding certain infinite algebras that allow easy testing[6] of φ to $\mathbf{A}$.

Corollary 17. *[Prototype algebras] Let Θ be a set of formulas, let $\mathbf{A}$ be a finite class of finite admitted Σ-algebras such that $\mathbf{A}$ sat Θ and $\Theta \models \mathcal{AX}_{\boldsymbol{x}}(\mathbf{A})$. Then, $\mathbf{A}$ is a class of prototype algebras for Θ and $\boldsymbol{x}$. Formally, for any formula φ over Σ and $\boldsymbol{x}$ we have $\Theta \models \varphi$ iff $\mathbf{A}$ sat φ.*

Proof.

"⇒":	trivial: $\mathbf{A}$ sat $\Theta \models \varphi$		
"⇐":	$\mathbf{A}$ sat φ		
	$\Rightarrow \mathcal{AX}_V(\mathbf{A}) \models \varphi$		by Thm. 13
	$\Rightarrow \Theta \models \varphi$		since $\Theta \models \mathcal{AX}_V(\mathbf{A})$

Note that the Corollary holds for arbitrary $\mathcal{S}_{\text{fix}}$ and $\mathcal{A}_{\text{fix}}$, not only for $Bool$. However, we need to fix $Bool$ in order to get the notion of $\models$ used in theorem proving. □

[6] In an equational setting, the singleton class containing the initial algebra is always a prototype. However, equality in the initial algebra is generally undecidable.

Example 18. Let Θ consist of the axioms for an Abelian group of characteristic 2. Referring to Exm. 14, we can see that Θ implies $\mathcal{AX}_{\langle x,y\rangle}(\mathcal{A}_2)$. Hence, in order to prove a formula φ over $\{x, y, 0, (+), (=)\}$ to be a consequence of Θ, it is sufficient by Cor. 17 just to test it in $\mathcal{A}_2$.

Unfortunately, we cannot get rid of "characteristic" equations: In any finite algebra with an associative binary operation $(+)$, a law $n_1x = n_2x$ holds for some $n_1 > n_2$, where we abbreviate $nx := x + \ldots + x$ (n times). Hence, each axiom set obtained from finitely many of such algebras necessarily entails a law of this form. □

Acknowledgements. Ingo Dahn drew our attention to the application area of prototype algebras in automated theorem proving. Martin Simons provided the literature reference to the many-sorted version of Birkhoff's variety theorem. Angela Sodan gave us some valuable advice on the presentation.

References

[AM91] A. Aiken and B. Murphy. Implementing regular tree expressions. In *ACM Conference on Functional Programming Languages and Computer Architecture*, pages 427–447, August 1991.

[BB91] J.M. Barzdin and G.J. Barzdin. Rapid construction of algebraic axioms from samples. *Theoretical Computer Science*, 90:199–208, 1991.

[BH96] Jochen Burghardt and Birgit Heinz. Implementing anti-unification modulo equational theory. Arbeitspapier 1006, GMD, Jun 1996.

[Bur02] Jochen Burghardt. Axiomatization of finite algebras. Arbeitspapier, GMD, 2002. forthcoming.

[CDG+99] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. WWW, Available from `www.grappa.univ-lille3.fr/tata`, Oct 1999.

[DJ90] N. Dershowitz and J.-P. Jouannaud. *Rewrite Systems*, volume B of *Handbook of Theoretical Computer Science*, pages 243–320. Elsevier, 1990.

[Gra94] Peter Graf. Substitution tree indexing. Technical Report MPI-I-94-251, Max-Planck-Institut für Informatik, Saarbrücken, Oct 1994.

[McA92] David McAllester. Grammar rewriting. In *Proc. CADE-11*, volume 607 of *LNAI*. Springer, 1992.

[MT92] K. Meinke and J.V. Tucker. *Universal Algebra*, volume 1 of *Handbook of Logic in Computer Science*. Clarendon, Oxford, 1992.

Algorithms for Guiding Clausal Temporal Resolution*

M. Carmen Fernández Gago, Michael Fisher, and Clare Dixon

Department of Computer Science, University of Liverpool
L69 7 ZF, United Kingdom
{M.C.Gago,M.Fisher,C.Dixon}@csc.liv.ac.uk

Abstract. Clausal temporal resolution is characterised by a *translation* of the formulae whose satisfiability is to be established to a normal form, *step* resolution (similar to classical resolution) on formulae occurring at the same states and *temporal* resolution between formulae describing properties over a longer period. The most complex part of the method occurs in searching for candidates for the temporal resolution operation, something that may need to be carried out several times.
In this paper we consider a new technique for finding the candidates for the temporal resolution operation. Although related to the previously developed external search procedure, this new approach not only allows the temporal resolution operation to be carried out at *any* moment, but also simplifies any subsequent search required for similar temporal formulae.
Finally, in contrast with previous approaches, this search can be seen as an inherent part of the resolution process, rather than an external procedure that is only called in certain situations.

1 Introduction

The effective mechanisation of temporal logic is vital to the application of temporal reasoning in many fields, for example the verification of reactive systems [12], the implementation of temporal query languages [4], and temporal logic programming [1]. Consequently, a range of proof methods have been developed, implemented and applied. The development of proof methods for temporal logic has followed three main approaches: tableau [16], automates [14] and resolution [2,3,10,15], the approach adopted here. Resolution based methods have the advantage that, as in the classical case [13], a range of strategies can be used.

A particularly successful strategy for classical resolution has been the set of support strategy [17], which restricts the application of the resolution rule, pruning the search space. Our aim is to develop a set of support (SOS) strategy for propositional temporal logic, PTL, the logic used in this paper. The extension of the SOS strategy for the fragment of PTL without eventualities (clauses involving the operator '$\Diamond$', meaning sometime in the future) has been achieved

* This work was partially supported by EPSRC grant GR/M44859.

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 235–249, 2002.

[7] using techniques developed from SOS for classical logic. The definition of the strategy for full PTL is non trivial and we intend to achieve it using the algorithms proposed in this paper together with the strategy defined for the case without eventualities.

Clausal temporal resolution [10] is characterised by the translation to a normal form, the application of classical style resolution between formulae that occur at the same moment in time (*step resolution*), together with a novel *temporal resolution* rule, which derives contradictions over temporal sequences. Although the clausal temporal resolution method has been defined, proved correct and implemented, it sometimes generates an unnecessarily large set of formulas that may be irrelevant to the refutation. Not only that, but temporal resolution operations occur only after many step resolution inferences have been carried out. This means that, in cases where a large amount of step resolution can occur, the method may be very expensive.

As the search for the candidates for the temporal resolution operation is the most expensive part of the method we need to guide it and, if possible, avoid much unnecessary subsequent step resolution. In this sense, we propose an algorithm based on step resolution to guide the search. In this approach, we choose a candidate formula for the temporal resolution operation and we check whether such a candidate is appropriated to perform the resolution operation. Our intention is to re-use as much information as possible in those cases where further searches are required. Thus, we propose a second algorithm which is based on the original one and is used to guide further searches. The structure of the paper is as follows. In Sect. 2 we define the temporal logic considered, namely Propositional Temporal Logic [11]. In Sect. 3 we review the basic resolution method. In Sect. 4 we describe an algorithm to find candidates for the temporal resolution operation using only step resolution. Its completeness is shown in Sect. 5. In Sect. 6 we propose a second algorithm algorithm for the cases when further searches are needed. Completeness is also shown in this section.

2 Syntax and Semantics of PTL

In this section we present the syntax and semantics of (PTL), based on a discrete, linear temporal logic with finite past and infinite future. The future-time connectives that we use include '$\Diamond$' (*sometime in the future*), '$\bigcirc$' (*in the next moment in time*), '$\Box$' (*always*) '$\mathcal{U}$' (*until*), and '$\mathcal{W}$' (*unless, or weak until*). A choice for interpreting such temporal connectives is $(\mathbb{N}, <)$, i.e., the Natural Numbers ordered by the usual 'less than' relation.

2.1 Syntax

PTL formulae are constructed using the following connectives and proposition symbols.

– A set, $\mathcal{P}$, of propositional symbols.

- Nullary connectives: **true** and **false**.
- Propositional connectives: $\neg$, $\vee$, $\wedge$, $\Rightarrow$ and $\Leftrightarrow$.
- Temporal connectives: $\bigcirc$, $\diamondsuit$, $\Box$, $\mathcal{U}$, and $\mathcal{W}$ and the nullary temporal connective **start**.

The set of well-formed formulae of PTL[1], denoted by WFF_p, is defined as the set satisfying:

- Any element of $\mathcal{P}$ is in WFF_p.
- **true**, **false** and **start** are in WFF_p.
- If ϕ and ψ are in WFF_p then so are $\neg\phi$, $\phi \vee \psi$, $\phi \wedge \psi$, $\phi \Rightarrow \psi$, $\phi \Leftrightarrow \psi$, $\diamondsuit\phi$, $\Box\phi$, $\phi \mathcal{U} \psi$, $\phi \mathcal{W} \psi$, $\bigcirc\phi$

2.2 Semantics

We define a model, M, for PTL as a structure $\langle \mathcal{D}, R, \pi_p \rangle$ where

- $\mathcal{D}$ is the temporal domain, e.g, the natural numbers and
- R is the ordering relation, e.g. $<$.
- $\pi_p : \mathcal{D} \times \mathcal{P} \rightarrow \{T, F\}$ is a function assigning T or F to each atomic proposition at each moment in time.

As usual we define the semantics of the language via the satisfaction relation '$\models$'. For PTL, this relation holds between pairs of the form $\langle M, u \rangle$ (M is a model and $u \in \mathbb{N}$) and well-formed formulae. The rules defining the satisfaction relation are as follows.

$\langle M, u \rangle \models p$ iff $\pi_p(u, p) = T$ (where $p \in \mathcal{P}$)
$\langle M, u \rangle \models$ **true**
$\langle M, u \rangle \not\models$ **false**
$\langle M, u \rangle \models$ **start** iff $u = 0$
$\langle M, u \rangle \models \phi \wedge \psi$ iff $\langle M, u \rangle \models \phi$ and $\langle M, u \rangle \models \psi$
$\langle M, u \rangle \models \phi \vee \psi$ iff $\langle M, u \rangle \models \phi$ or $\langle M, u \rangle \models \psi$
$\langle M, u \rangle \models \phi \Rightarrow \psi$ iff $\langle M, u \rangle \not\models \phi$ or $\langle M, u \rangle \models \psi$
$\langle M, u \rangle \models \neg\phi$ iff $\langle M, u \rangle \not\models \phi$
$\langle M, u \rangle \models \phi \Leftrightarrow \psi$ iff $\langle M, u \rangle \models \phi \Rightarrow \psi$ and $\langle M, u \rangle \models \psi \Rightarrow \phi$
$\langle M, u \rangle \models \bigcirc\phi$ iff $\langle M, u + 1 \rangle \models \phi$
$\langle M, u \rangle \models \diamondsuit\phi$ iff there exists a $k \in \mathbb{N}$ such that $k \geq u$ $\langle M, k \rangle \models \phi$
$\langle M, u \rangle \models \Box\phi$ iff for all $j \in \mathbb{N}$, if $j \geq u$ then $\langle M, j \rangle \models \phi$
$\langle M, u \rangle \models \phi \mathcal{U} \psi$ iff there exists a $k \in \mathbb{N}$, s.t. $k \geq u$ and $\langle M, k \rangle \models \psi$ and
for all $j \in \mathbb{N}$, if $u \leq j < k$ then $\langle M, j \rangle \models \phi$
$\langle M, u \rangle \models \phi \mathcal{W} \psi$ iff $\langle M, u \rangle \models \phi \mathcal{U} \psi$ or $\langle M, u \rangle \models \Box\phi$

[1] As usual, parentheses are also allowed to avoid ambiguity

3 Clausal Resolution Method for PTL

The resolution method presented here is clausal, that means that to assure the validity of some PTL formula we negate it and translate into a normal form. Then, both step resolution and temporal resolution are applied. We terminate when either a contradiction has been derived or no new information can be derived.

3.1 Separated Normal Form

The resolution method depends on formulae being transformed into a normal form (SNF). The normal form, which is presented in [9], comprises formulae that are implications with present-time formulae on the left-hand side and (present or) future-time formulae on the right-hand-side. The transformation of formulae into SNF depends on three main operations: the renaming of complex subformulae; the removal of temporal operators; and classical style rewrite operations. In this section we review SNF but do not consider the transformation procedure (we note that the transformation to SNF preserves satisfiability [10]).

Formulae in SNF are of the general form $\Box \bigwedge_i (\phi_i \Rightarrow \psi_i)$, where each $\phi_i \Rightarrow \psi_i$ is known as a *clause* and is one of the following forms

$$\begin{array}{ll} \mathbf{start} \Rightarrow \bigvee_c l_c & \text{(an } \textit{initial} \text{ clause)} \\ \bigwedge_a k_a \Rightarrow \bigcirc \bigvee_d l_d & \text{(a } \textit{step} \text{ clause)} \\ \bigwedge_b k_b \Rightarrow \Diamond l & \text{(a } \textit{sometime} \text{ clause)} \end{array}$$

where each k_a, k_b, l_c, l_d and l represent literals.

To apply the temporal resolution operation described below, one or more step clauses may need to be combined. Then a variant on SNF called *merged-SNF* (*SNF*$_m$)[8] is also defined. Given a set of clauses in SNF, any clause in SNF is also a clause in SNF$_m$. Any two clauses in SNF$_m$ may be combined to produce a clause in SNF$_m$ as follows.

$$\frac{\begin{array}{c}\phi_1 \Rightarrow \bigcirc \psi_1 \\ \phi_2 \Rightarrow \bigcirc \psi_2\end{array}}{(\phi_1 \wedge \phi_2) \Rightarrow \bigcirc (\psi_1 \wedge \phi_2)}$$

3.2 Resolution Operations

Step resolution consists of the application of the standard classical resolution rule in two different contexts. Pairs of initial or step clauses may be resolved as follows:

$$\frac{\begin{array}{l}\mathbf{start} \Rightarrow \psi_1 \vee l \\ \mathbf{start} \Rightarrow \psi_2 \vee \neg l\end{array}}{\mathbf{start} \Rightarrow \psi_1 \vee \psi_2} \qquad \frac{\begin{array}{r}\phi_1 \Rightarrow \bigcirc (\psi_1 \vee l) \\ \phi_2 \Rightarrow \bigcirc (\psi_2 \vee \neg l)\end{array}}{(\phi_1 \wedge \phi_2) \Rightarrow \bigcirc (\psi_1 \vee \psi_2)}$$

The *simplification operations* are similar to those used in the classical case, consisting of both simplification and subsumption. An additional operation is required when a temporal contradiction is produced:

$$\frac{\phi \Rightarrow \bigcirc \mathbf{false}}{\begin{array}{l}\mathbf{start} \Rightarrow \neg\phi \\ \mathbf{true} \Rightarrow \bigcirc\neg\phi\end{array}}$$

This means that, if a formula ϕ leads to a contradiction in the next moment, then ϕ must never be satisfied.
Temporal resolution operations resolve one sometime clause with a set of merged step clauses [10] as follows:

$$\frac{\begin{array}{c}\phi_1 \Rightarrow \bigcirc\psi_1 \\ \vdots \quad \vdots \quad \vdots \\ \phi_n \Rightarrow \bigcirc\psi_n \\ \chi \Rightarrow \diamondsuit\neg l\end{array}}{\chi \Rightarrow (\neg \bigvee_{i=1}^{n} \phi_i)\mathcal{W}\neg l}$$

with the side condition that for all i, $1 \leq i \leq n$, then $\models \psi_i \Rightarrow l$ and $\models \psi_i \Rightarrow \bigvee_{j=1}^{n} \phi_j$, from which we can derive $\bigwedge_{i=1}^{n}(\phi_i \Rightarrow \bigcirc(l \wedge \bigvee_{j=1}^{n} \phi_j))$. This side condition ensures that the set of $\phi_i \Rightarrow \bigcirc\psi_i$ merged clauses together imply $\bigvee_{i=1}^{n} \phi_i \Rightarrow \bigcirc\,\Box l$. Such a set of clauses is known as a *loop* in l. The resolvent produced includes an $\mathcal{W}$ operator that must be translated into SNF before any further resolution steps can be applied.

Termination. If **start**⇒ **false** is produced, the original formula is unsatisfiable and the resolution process terminates.

Correctness. The soundness and (refutation) completeness of the original temporal resolution method have been both established in [10].

4 Algorithm for Searching for Loops

In order to apply the resolution rule presented in Sect. 3 a loop must be detected. Thus, given an eventuality $\diamondsuit\neg l$, our aim is to detect a set of merged step clauses that comprises a loop to be resolved with $\diamondsuit\neg l$.

4.1 Motivation

In [5] a breadth-search approach is used to detect loops. Although this algorithm is correct, in some cases, when further searches for loops need to be carried out, the information obtained in a previous search is not reused. Our approach here is based on step resolution and allows us to re-use previous search information. Assume we are searching for a loop in l, our search produces a sequence of *guesses*, G_i, which are DNF formulae. We show that these are equivalent to the DNF formulae H_i output by the Breadth-First Search algorithm (see [5]). In Breadth-First Search each new DNF formula H_{i+1} satisfies the property $H_{i+1} \Rightarrow \bigcirc(H_i \wedge l)$. Similarly we also have $G_{i+1} \Rightarrow \bigcirc(G_i \wedge l)$. In order to find G_{i+1} we add $\mathbf{true} \Rightarrow \bigcirc(\neg G_i \vee \neg l)$ to the original set of clauses and resolve. The left hand side of clauses $Z \Rightarrow \bigcirc\mathbf{false}$ satisfy $Z \Rightarrow \bigcirc(G_i \wedge l)$. As we want to save clauses derived during this process and possibly use them later, we add the clause $s_i^{\neg l} \Rightarrow \bigcirc(\neg G_i \vee \neg l)$ and thus search for clauses $s_i^{\neg l} \wedge Z \Rightarrow \bigcirc\mathbf{false}$, derived from resolving with $s_i^{\neg l} \Rightarrow \bigcirc(\neg G_i \vee \neg l)$ or its resolvents with other clauses which are rewritten as $\mathbf{true} \Rightarrow \bigcirc(\neg s_i^{\neg l} \vee \neg Z)$.

4.2 Step Loop Search Algorithm

In this section we propose an algorithm to search for a loop. For each eventuality $\diamondsuit\neg l$ occurring on the right hand side of a sometime clause, the algorithm constructs a sequence of DNF formulae, G_i, by using the previous guess together with F_j, where F_j are disjunctions of literals derived by the application of the algorithm to G_i. The algorithm is the following.

1. Choose $G_{-1} \Leftrightarrow \mathbf{true}$
2. Given a guess G_i add the clause $s_i^{\neg l} \Rightarrow \bigcirc(\neg G_i \vee \neg l)$ and apply Step Resolution.
3. For all clauses $\mathbf{true} \Rightarrow \bigcirc(\neg s_i^{\neg l} \vee F_j)$ obtained during the generation of resolvents, let $G_{i+1} \Leftrightarrow G_i \wedge (\bigvee_{j=1}^{m} \neg F_j)$.
4. Go to 2 until either
   a) $G_i \Leftrightarrow G_{i+1}$ (we terminate having found a loop).
   b) G_{i+1} is empty. (we terminate without having found a loop).

4.3 Example

Let the loop be $(a \wedge b \wedge c \wedge d) \Rightarrow \bigcirc \Box l$, derived from the following SNF clauses.

1. $a \Rightarrow \bigcirc l$
2. $b \wedge c \Rightarrow \bigcirc d$
3. $c \wedge d \Rightarrow \bigcirc a$
4. $d \wedge a \Rightarrow \bigcirc b$
5. $a \wedge b \Rightarrow \bigcirc c$
6. $\chi \Rightarrow \diamondsuit\neg l$

According to algorithm 1 the first guess is $G_{-1} \Leftrightarrow \mathbf{true}$. For such a guess we add the clause $s_{-1}^{\neg l} \Rightarrow \bigcirc(\mathbf{false} \vee \neg l)$. Some of the resolvents derived by applying step resolution among this clause and 1-6 are

7. $s_{-1}^{\neg l} \Rightarrow \bigcirc \neg l$
8. $s_{-1}^{\neg l} \wedge a \Rightarrow \bigcirc \mathbf{false}$ [1, 7]
9. $\mathbf{true} \Rightarrow \bigcirc(\neg s_{-1}^{\neg l} \vee \neg a)$ [Simp.8]

Therefore, the next guess will be, $G_0 \Leftrightarrow \mathbf{true} \wedge a \Leftrightarrow a$.

10. $s_0^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg a)$
11. $s_0^{\neg l} \wedge a \wedge c \wedge d \Rightarrow \bigcirc \mathbf{false}$ [1, 3, 10]
12. $\mathbf{true} \Rightarrow \bigcirc(\neg s_0^{\neg l} \vee \neg a \vee \neg c \vee \neg d)$ [Simp.11]

Next guess is $G_1 \Leftrightarrow a \wedge (a \wedge c \wedge d) \Leftrightarrow a \wedge c \wedge d$.

13. $s_1^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg a \vee \neg c \vee \neg d)$
14. $s_1^{\neg l} \wedge a \wedge b \wedge c \wedge d \Rightarrow \bigcirc \mathbf{false}$ [1, 2, 3, 5, 13]
15. $\mathbf{true} \Rightarrow \bigcirc(\neg s_1^{\neg l} \vee \neg a \vee \neg b \vee \neg c \vee \neg d)$ [Simp.14]

According to the algorithm the next guess is

$$G_2 \Leftrightarrow (a \wedge c \wedge d) \wedge (a \wedge b \wedge c \wedge d) \Leftrightarrow a \wedge c \wedge d \wedge b$$

16. $s_2^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg a \vee \neg c \vee \neg d \vee \neg b)$
17. $s_2^{\neg l} \wedge a \wedge b \wedge c \wedge d \Rightarrow \bigcirc \mathbf{false}$ [1, 2, 3, 4, 5, 16]
18. $\mathbf{true} \Rightarrow \bigcirc(\neg s_2^{\neg l} \vee \neg a \vee \neg b \vee \neg c \vee \neg d)$ [Simp.17]

If we apply the algorithm to G_2 then the next guess will again be

$G_3 \Leftrightarrow (a \wedge b \wedge c \wedge d) \wedge (a \wedge b \wedge c \wedge d) \Leftrightarrow (a \wedge b \wedge c \wedge d)$.
$G_3 \Leftrightarrow G_2$

which means termination as $G_3 \Leftrightarrow G_2$ and so G_2 is a loop, i.e, $G_2 \Rightarrow \bigcirc \Box l$.

5 Completeness

In the following we will prove completeness for this algorithm by relating it to the completeness of the Breadth-First Search Algorithm [5]. We first introduce the Breadth-First Search Algorithm.

5.1 Breadth-First Search Algorithm

The Breadth-First Search Algorithm constructs a sequence of formulae, H_i for $i \geq 0$, that are formulae in Disjunctive Normal Form and contain no temporal operators. They are constructed from the conjunctions of literals on the left hand sides of step clauses or combinations of step clauses in the SNF-clause-set that satisfy certain properties (see below). Assuming we are resolving with $\Diamond \neg l$ each

formula H_i satisfies $H_i \Rightarrow \bigcirc l$ and given H_i each new formula H_{i+1} satisfies $H_{i+1} \Rightarrow \bigcirc H_i$ and $H_{i+1} \Rightarrow H_i$. When termination occurs we have $H_{i+1} \Leftrightarrow H_i$ so that $H_i \Rightarrow \bigcirc \Box l$ for resolution with $\diamondsuit \neg l$. The algorithm assumes that all necessary step resolution has been carried out.

Breadth-First Search Algorithm. For each eventuality $\diamondsuit \neg l$ occurring on the right hand side of a sometime clause do the following.

1. Search for all the step clauses of the form $C_k \Rightarrow \bigcirc l$, for $k = 0$ to b, disjoin the left hand sides and generate the H_0 equivalent to this, i.e. $H_0 \Leftrightarrow \bigvee_{k=0}^{b} C_k$. Simplify H_0. If $\models H_0$ we terminate having found a loop-formula (**true**).
2. Given formula H_i, build formula H_{i+1} for $i = 0, 1, \ldots$ by looking for step clauses or combinations of clauses of the form $A_j \Rightarrow \bigcirc B_j$, for $j = 0$ to c where $\models B_j \Rightarrow H_i$ and $\models A_j \Rightarrow H_0$. Disjoin the left hand sides so that $H_{i+1} \Leftrightarrow \bigvee_{j=0}^{c} A_j$ and simplify as previously.
3. Repeat (2) until
   a) $\models H_i$. We terminate having found a loop-formula and return **true**.
   b) $\models H_i \Leftrightarrow H_{i+1}$. We terminate having found a loop-formula and return the DNF formula H_i.
   c) The new formula is empty. We terminate without having found a loop-formula.

Soundness, Completeness, and Termination for the BFS-Algorithm [5]. Given a set of SNF clauses R, that contains a loop $A \Rightarrow \bigcirc \Box l$, applying BFS algorithm will output a DNF formula A' such that $A' \Rightarrow \bigcirc \Box l$ and $A \Rightarrow A'$. Termination of the BFS algorithm is also established.[5]

5.2 Completeness of the Step Loop Search Algorithm

To show the completeness of the new algorithm we will prove that for all $i \geq 0$, $G_i \Leftrightarrow H_i$ by induction. Let R be a set of SNF-clauses and $\diamondsuit \neg l$ be the right hand side of a sometime clause, we assume that R contains a loop in l.

Lemma 1. $G_0 \Leftrightarrow H_0$

Proof. In order to obtain G_0, according to the algorithm, the clause $s_{-1}^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg \mathbf{true})$ is added, which is the clause $s_{-1}^{\neg l} \Rightarrow \bigcirc \neg l$ (1).
As R contains a loop, in the initial set of clauses there must be some clauses such that they may be resolved together to obtain $A_i \Rightarrow \bigcirc l$, $1 \leq i \leq k$.
By resolution with clause (1) the resolvents are $s_{-1}^{\neg l} \wedge A_i \Rightarrow \bigcirc \mathbf{false}$, $1 \leq i \leq k$ and by simplification $\mathbf{true} \Rightarrow \bigcirc(\neg s_{-1}^{\neg l} \vee \neg A_i)$.
These last clauses are used in order to obtain G_0 as $G_0 \Leftrightarrow \mathbf{true} \wedge (A_1 \vee \ldots \vee A_k) \Leftrightarrow A_1 \vee \ldots \vee A_k$.

For building H_0 by the Breadth-First Search Algorithm, the left hand sides of the clauses $A_i \Rightarrow \bigcirc l$ are disjoined, giving , $H_0 \Leftrightarrow A_1 \vee ... \vee A_k$, and so, $G_0 \Leftrightarrow H_0$ □

Theorem 1. *For all* $n \in \mathbb{N}$ $H_n \Leftrightarrow G_n$.

Proof. Base case: By Lemma 1, $H_0 \Leftrightarrow G_0$.
Induction case: We assume $H_k \Leftrightarrow G_k$ for all $k \leq i$, $k \in \mathbb{N}$ and we prove the hypothesis for $i+1$. We know the following valid statements about H_{i+1} from the definition of the Breadth First Search algorithm:

$$\begin{array}{ll} (a).\ H_{i+1} \Rightarrow & \bigcirc H_i \\ (b).\ H_{i+1} \Rightarrow & \bigcirc l \\ (c).\ \ G_i \ \Leftrightarrow & H_i \quad \text{(Induction Hypothesis)} \\ (d).\ H_{i+1} \Rightarrow & H_i \end{array}$$

Assume we have generated guess G_i and we are about to derive G_{i+1}. From the algorithm the clause $s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg G_i)$ is added. Using property (c) the clause 1 is transformed into

$$s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg H_i). \tag{1}$$

Then, applying step resolution, we obtain:

$$\begin{array}{lll} 3.\ s_i^{\neg l} \wedge H_{i+1} \Rightarrow \bigcirc \neg H_i & & [\text{b, 1}] \\ 4.\ s_i^{\neg l} \wedge H_{i+1} \Rightarrow \bigcirc \mathbf{false} & & [3,\text{a}] \\ 5. \quad \mathbf{true} \quad \Rightarrow \bigcirc(\neg s_i^{\neg l} \vee \neg H_{i+1}) & & [\text{Simp. 4}] \end{array}$$

In order to obtain G_{i+1} the algorithm is applied, where the clauses
$\mathbf{true} \Rightarrow \bigcirc(\neg s_i^{\neg l} \vee F_j)$ considered in this case just consist of clause 5 and thus
$G_{i+1} \Leftrightarrow G_i \wedge [H_{i+1}] \Leftrightarrow G_i \wedge H_{i+1} \Leftrightarrow H_i \wedge H_{i+1}$.
By property (d) $(H_i \wedge H_{i+1}) \Leftrightarrow H_{i+1}$, and then $G_{i+1} \Leftrightarrow H_{i+1}$ □

Theorem 2. *If* G_i *is a loop, then* $G_i \Leftrightarrow G_{i+1}$.

Proof. Let $G_i \Leftrightarrow D_1 \vee D_2 \vee ... \vee D_n$ be a loop.
As usual for the application of the algorithm the clause

$$s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg G_i) \tag{2}$$

is added.

As G_i is a loop, there must be a set of clauses in the initial set of clauses that together represent $G_i \Rightarrow \bigcirc l$ and $G_i \Rightarrow \bigcirc G_i$.

Resolution can be applied among these clauses and clause 2 producing
$s_i^{\neg l} \wedge G_i \Rightarrow \bigcirc \mathbf{false}$. By applying simplification this latter clause is transformed into $\mathbf{true} \Rightarrow \bigcirc(\neg s_i^{\neg l} \vee \neg G_i)$. Then, $G_{i+1} \Leftrightarrow G_i \wedge G_i$ and so $G_{i+1} \Leftrightarrow G_i$. □

Theorem 3. *The Step Loop Search Algorithm is complete.*

Proof. By Theorem 1 and 2.

□

Theorem 4. *The algorithm terminates.*

Proof. 1. If there exists a loop in the initial set of clauses, then we know that Breadth Search Algorithm terminates finding a loop H_n. By Theorem 1, $H_n \Leftrightarrow G_n$, so G_n is a loop and then by Theorem 2 $G_n \Leftrightarrow G_{n+1}$. Thus, the algorithm terminates by 2 *(a)*.
2. If there does not exist a set of clauses which comprise a loop in the initial set of clauses then, at some point, it will not be possible to produce the clause $s_i^{\neg l} \wedge C_i \Rightarrow \bigcirc \mathbf{false}$ and therefore no clauses $\mathbf{true} \Rightarrow \bigcirc(\neg s_i^{\neg l} \vee F_i)$ will be derived during the proof and G_{i+1} will be empty.

□

6 The Search for a Subsequent Loop

We assume that we have detected a sequence of guesses by applying the previous algorithm to a set of SNF-clauses, X, with an eventuality $\Diamond\neg l$ but, later we need to apply temporal resolution again to $\Diamond\neg l$. Further, the application of the temporal resolution rule to this or other eventualities has led to the generation of new clauses which we may use in order to generate a new loop. Let Y be the set of new SNF-clauses generated. Thus the set of SNF-clauses is now updated as $X \cup Y$.

Rather than carrying out the full loop search again, our intention is to re-use the original loop search, even though a loop may not have been detected in the first search.

6.1 Repeated Step Loop Search Algorithm

Assume we have $G_0, ..., G_n$ guesses from a previous search for a loop and, the added set of clauses is Y. We can generate a new sequence of DNF formula, K_i as follows:

1. Guess $K_{-1} \Leftrightarrow \mathbf{false}$.
2. Given a guess K_i add the clause $s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg K_i)$ to the piece of proof of the previous search corresponding with s_i, together with the clauses Y.
3. Apply Step Resolution.
4. For all new clauses $\mathbf{true} \Rightarrow \bigcirc(\neg s_i^{\neg l} \vee F'_j)$ obtained during the proof, let
   $K_{i+1} \Leftrightarrow (G_i \vee K_i) \wedge (\bigvee_{j=1}^{m} \neg F'_j)$
5. Go to 2 until either
   a) $K_i \Rightarrow K_{i+1} \vee G_{i+1}$ where $K_i \not= \mathbf{false}$ or
   b) $K_i \vee G_i \Leftrightarrow K_{i+1} \vee G_{i+1}$,
   whatever is the earliest.

6.2 Example

We now consider the example in Sect. 4.3 where clause 4 has been deleted. In this case, all the guesses remain the same except the last one which now will be the following

16. $s_2^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg a \vee \neg c \vee \neg d \vee \neg b)$
17. $s_2^{\neg l} \wedge a \wedge b \wedge c \wedge d \Rightarrow \bigcirc \neg b$ [1, 2, 3, 5, 16]

As there are not clauses $\mathbf{true} \Rightarrow \bigcirc(\neg s_2^{\neg l} \vee ....)$, then $G_3 \Leftrightarrow \mathbf{false}$.
Thus, the search for a loop failed as the set of clauses did not comprise a loop but we still can use these previous guesses in order to find a loop if the appropriate clause is later added.
Now we consider the same example but we add clause 4, that is, $d \wedge a \Rightarrow \bigcirc b$.
Algorithm 2 is now applied, with $K_{-1} \Leftrightarrow \mathbf{false}$. The new clause added is

18. $s_{-1}^{\neg l} \Rightarrow \bigcirc \mathbf{true}$

Nothing new is added and no clauses from 7-9 in example 4.3 can be resolved with 4. Thus, $K_0 \Leftrightarrow \mathbf{false}$ and $T_0 \Leftrightarrow a \vee \mathbf{false} \Leftrightarrow a$.

19. $s_0^{\neg l} \Rightarrow \bigcirc \mathbf{true}$

Again nothing new is added and no clauses from 10-12 in the example 4.3 can be resolved with 4. Thus, $K_1 \Leftrightarrow \mathbf{false}$ and $T_1 \Leftrightarrow (a \wedge b) \vee \mathbf{false} \Leftrightarrow (a \wedge b)$.

20. $s_1^{\neg l} \Rightarrow \bigcirc \mathbf{true}$

As happened in the previous cases no new clauses are added from resolution between 13-15 and 4, so $K_2 \Leftrightarrow \mathbf{false}$ and $T_2 \Leftrightarrow (a \wedge b \wedge c \wedge d) \vee \mathbf{false} \Leftrightarrow a \wedge b \wedge c \wedge d$.

20. $s_2^{\neg l} \Rightarrow \bigcirc \mathbf{true}$
21. $s_2^{\neg l} \wedge a \wedge b \wedge c \wedge d \Rightarrow \bigcirc \mathbf{false}$ [17, 4]
22. $\mathbf{true} \Rightarrow \bigcirc(\neg s_2^{\neg l} \vee \neg a \vee \neg b \vee \neg c \vee \neg d)$ [Simp.22]

$K_3 \Leftrightarrow (a \wedge b \wedge c \wedge d) \wedge (a \wedge b \wedge c \wedge d) \Leftrightarrow (a \wedge b \wedge c \wedge d)$.
$T_3 \Leftrightarrow (a \wedge b \wedge c \wedge d)$.
$T_2 \Leftrightarrow T_3$. Then T_3 is a loop.

6.3 The New Guess

Let $G_{-1}, G_0, ..., G_j$ be the guesses generated by applying Algorithm 1 to a set of clauses X. Let Y be the new set of clauses added to X and $T_{-1}, T_0, ..., T_i$ be the guesses obtained by applying Algorithm 1 to $X \cup Y$. $K_{-1}, K_0, ..., K_p$ are the guesses obtained by applying Algorithm 2.

Theorem 5. *The new guess T_i, such that T_i / $\Leftrightarrow$ G_i, K_i / $\Leftrightarrow$ **false** has the property $T_i \Leftrightarrow G_i \vee K_i$.*

Proof. We assume that the new guess is generated from the fragment of proof corresponding to $s_{i-1}^{\neg l}$ and $K_i \not\mathit{imp}\mathbf{false}$ whereas $K_{-1}, K_0, ..., K_{i-1} \Leftrightarrow \mathbf{false}$.
Let $\mathbf{true} \Rightarrow \bigcirc(\neg s_{i-1}^{\neg l} \vee F_j)$ be the clauses used for Algorithm 1 in order to generate G_i.
As a result of adding the new clauses Y, if step resolution is applied among these clauses, X or those containing s_{i-1} on the left hand side, clauses
$\mathbf{true} \Rightarrow \bigcirc(\neg s_{i-1}^{\neg l} \vee C_k)$ may be generated, where $F_j \not\Leftrightarrow C_k$. N.B. that C_k must exist as we are assuming $K_i \not\Leftrightarrow \mathbf{false}$.
By applying algorithm 1, the new guess will be

$$\begin{aligned} T_i &\Leftrightarrow G_{i-1} \wedge [(\bigvee_{j=1}^{m} \neg F_j) \vee (\bigvee_{k=1}^{p} \neg C_k)] \\ &\Leftrightarrow (G_{i-1} \wedge (\bigvee_{j=1}^{m} \neg F_j)) \vee (G_{i-1} \wedge (\bigvee_{k=1}^{p} \neg C_k)) \\ &\Leftrightarrow G_i \vee K_i. \end{aligned}$$

□

We have shown that the first new guess is like $T_i \Leftrightarrow G_i \vee K_i$. If $K_i \not\Leftrightarrow \mathbf{false}$ then trivially $T_i \Leftrightarrow G_i \vee \mathbf{false}$. Next we will show that this is the case for all the new guesses.

Theorem 6. *Let i the first index such that $T_i \not\Leftrightarrow G_i$. Then for all j, $j \geq i$, $T_j \Leftrightarrow G_j \vee K_j$.*

Proof. We assume that $T_j \Leftrightarrow G_j \vee K_j$. We will prove that $T_{j+1} \Leftrightarrow G_{j+1} \vee K_{j+1}$. In order to obtain T_{j+1} the original algorithm must be applied to T_j. Thus, the clause added is $s_j^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg T_j)$. As we know that $T_j \Leftrightarrow G_j \vee K_j$ the previous clause can be rewritten as

$s_j^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg G_j)$ (1)
$s_j^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg K_j)$ (2)

As clause (1) was produced while obtaining G_{j+1}, the clauses
$\mathbf{true} \Rightarrow \bigcirc(\neg s_j^{\neg l} \vee F_{j+1})$ that appeared in the original search will be generated again. From clause (2) and $X \cup Y$ new clauses $\mathbf{true} \Rightarrow \bigcirc(\neg s_j^{\neg l} \vee C_{k+1})$ may be derived. Then by applying algorithm 1.

$$\begin{aligned} T_{j+1} &\Leftrightarrow T_j \wedge [(\bigvee_{j=1}^{m} \neg F_{j+1}) \vee (\bigvee_{k=1}^{p} \neg C_{k+1})] \\ &\Leftrightarrow (G_j \vee K_j) \wedge [(\bigvee_{j=1}^{m} \neg F_{j+1}) \vee (\bigvee_{k=1}^{p} \neg C_{k+1})] \\ &\Leftrightarrow [G_j \wedge (\bigvee_{j=1}^{m} \neg F_{j+1})] \vee [K_j \wedge (\bigvee_{j=1}^{m} \neg F_{j+1})] \vee [(G_j \vee K_j) \wedge (\bigvee_{j=1}^{m} \neg C_{k+1})] \\ &\Leftrightarrow G_{j+1} \vee K_{j+1} \end{aligned}$$

As $K_j \wedge (\bigvee_{j=1}^{m} \neg F_{j+1})$ is subsumed by $(G_j \vee K_j) \wedge (\bigvee_{j=1}^{m} \neg C_{k+1})$. □

6.4 Completeness of the Repeated Step Loop Search Algorithm

Theorem 7. *Algorithm 2 is complete.*

Proof. The proof follows Theorem 5 and Theorem 6 and completeness of the Step Loop Search Algorithm. □

Termination

Lemma 2. *For all i,* $K_{i+1} \Rightarrow K_i \vee G_i$.

Proof. Let $T_{-1}, T_0, ..., T_{i+1}$ be the guesses from applying Algorithm 1 to the clauses $X \cup Y$.
By Theorem 1 and definition of Breadth-First Search algorithm, $T_{i+1} \Rightarrow T_i$.
By Theorem 6 $T_i \Leftrightarrow G_i \vee K_i$. Then $K_{i+1} \vee G_{i+1} \Rightarrow K_i \vee G_i$. Therefore, $K_{i+1} \Rightarrow K_i \vee G_i$. □

Theorem 8. *The algorithm for searching for* K_i *terminates.*

Proof. We know that the search for T_i must terminate because of termination of the algorithm described in Sect. 4.2 when $T_i \Leftrightarrow T_{i+1}$. By Lemma 2 we know that $K_{i+1} \Rightarrow K_i \vee G_i$ if $K_i \not\Leftrightarrow$ **false**. K_i has the property that $T_i \Leftrightarrow G_i \vee K_i$. Then $G_i \vee K_i \Leftrightarrow G_{i+1} \vee K_{i+1}$. Therefore $K_i \Rightarrow G_{i+1} \vee K_{i+1}$. □

7 Some Advantages of Algorithm 2

Let X be a set of SNF-clauses with an eventuality $\Diamond\neg l$ when we apply Algorithm 1 to X, we obtain a sequence of guesses $G_{-1}, G_0, ..., G_n$.
Now we assume that some new set of SNF clauses, Y, have been added and we intend to search for a loop in the set of clauses $X \cup Y$.
As we have shown in Sect. 6.3, if a new set of clauses is added the new guess has the property $T_i \Leftrightarrow G_i \vee K_i$, where K_i is given by Algorithm 2. Applying the algorithm 1 to $X \cup Y$ for every new guess T_i, the clauses that we must add for generating the next guess are

$$s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg G_i)$$
$$s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg K_i)$$

Thus, all step resolution amongst clauses $s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg K_i)$ and the set of clauses X will be produced again, as they were produced when Algorithm 1 was applied just to X.

If we apply Algorithm 2 instead, we can save all those resolution steps as we only add clauses of the form $s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg K_i)$.

8 Conclusions and Future Work

In this paper we have presented two algorithms for searching for loops based upon step resolution.

The first algorithm uses outputs from the previous guess to guide the choice for the next guess and its correctness is shown with respect to an existing loop search algorithm. The second algorithm is based on the first one and allows us search for a second loop without having to carry out a whole search again, since it uses clauses generated during previous searches.

In the future we intend to apply these results to the development of strategies for temporal resolution that allows us to reduce the search space. In particular, we are interested in incorporating the set of support strategy [17,6]. In [7] the set of support is defined for the case without eventualities. For full temporal resolution the situation is more complex and we intend to achieve it combining and extending the results presented in [7] and the results in this paper.
Even though the algorithms presented are used in order to search for loops for the application of the temporal resolution operation, we can still guide this search. Thus, our intention is to define a set of support strategy for temporal resolution which involves a set of support for the search for loops process and then combines it with the set of support for step resolution. Thus, if we consider the example in Sect. 6.2 the Set of Support (SOS) for searching for a loop will include clauses of the form $s_i^{\neg l} \Rightarrow \bigcirc(\neg l \vee \neg K_i)$.

The practical efficiency of the algorithms is part of current work. It is expected to be examined while updating an existing implementation for temporal resolution where the search for a loop process is substituted with the algorithms presented here.

We also intend to investigate the detailed complexity of this approach.

References

1. M. Abadi and Z. Manna. Temporal Logic Programming. *Journal of Symbolic Computation*, 8: 277–295, 1989.
2. M. Abadi and Z. Manna. Nonclausal Deduction in First-Order Temporal Logic. *ACM Journal*, 37(2):279–317, April 1990.
3. A. Cavalli and L. Farinas del Cerro. A Decision Method for Linear Temporal Logic. In R.E.Shostak, editor, *Proceedings of the 7th International Conference on Automated Deduction*, volume 170 of *Lecture Notes in Computer Science*, pages 113–127. Springer-Verlag, 1984.
4. J. Chomicki and D. Niwinski. On the Feasibility of Checking Temporal Integrity Constraints. *Journal of Computer and System Sciences*, 51(3):523–535, December 1995.
5. C. Dixon. Temporal Resolution using a Breadth-First Search Algorithm. *Annals of Mathematics and Artificial Intelligence*, 22:87–115, 1998.
6. C. Dixon and M. Fisher. The Set of Support Strategy in Temporal Resolution. In *Proceedings of TIME-98 the Fifth International Workshop on Temporal Representation and Reasoning*, Sanibel Island, Florida, May 1998. IEEE Computer Society Press.

7. M.C. Fernández-Gago. Efficient Control of Temporal Reasoning. Transfer Report, Manchester Metropolitan University, 2000.
8. M. Fisher. A Resolution Method for Temporal Logic. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 99–104, Sydney, Australia, August 1991. Morgan Kaufman.
9. M. Fisher. A Normal Form for Temporal Logic and its Application in Theorem-Proving and Execution. *Journal of Logic and Computation*, 7(4):429–456, August 1997.
10. M. Fisher, C. Dixon, and M. Peim. Clausal Temporal Resolution. In *Transactions on Computational Logic 2(1)*, January 2001.
11. D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. The Temporal Analysis of Fairness. In *Proceedings of the 7th ACM Symposium on the Principles of Programming Languages*, pages 163–173, Las Vegas, Nevada, January 1980.
12. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1992.
13. J. A. Robinson. A Machine–Oriented Logic Based on the Resolution Principle. *ACM Journal*, 12(1):23–41, January 1965.
14. A. P. Sistla, M. Vardi, and P. Wolper. The Complementation Problem for Buchi Automata with Applications to Temporal Logic. *Theoretical Computer Science*, 49:217–237, 1987.
15. G. Venkatesh. A Decision Method for Temporal Logic based on Resolution. *Lecture Notes in Computer Science*, 206:272–289, 1986.
16. P. Wolper. The Tableau Method for Temporal Logic: An overview. *Logique et Analyse*, 110–111:119–136, June–Sept 1985.
17. L. Wos, G. Robinson, and D. Carson. Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. *ACM Journal*, 12:536–541, October 1965.

Axiom – A Modular Visual Object Retrieval System

Jochen Wickel, Pablo Alvarado, Peter Dörfler, Thomas Krüger, and Karl-Friedrich Kraiss

Lehrstuhl für Technische Informatik, RWTH Aachen
http://www.techinfo.rwth-aachen.de/

Abstract. Computer vision has always been an active research domain within artificial intelligence. Recognizing visual objects can alleviate the interaction of users with information retrieval systems. In this paper, we present a modular object recognition system which combines advanced image processing methods with AI techniques in a flexible way. This flexibility permits adaptations to a large variety of tasks. We describe the system architecture, point out some of the key algorithms and present experimental results which demonstrate the system's performance in several recognition tasks.

1 Introduction

The visual recognition of real world objects is an essential task of systems that are supposed to interact with their environment in a human-like way. In this paper, we present the modular object recognition system Axiom (Adaptive eXpert system for Intelligent Object Mining), which is composed of a collection of modules that can be combined in order to create computer vision and image processing applications.

The general application framework for our system is visual object retrieval. Suppose, for instance, we have some kind of object, like the cogwheel depicted in Fig. 1, and we need some information concerning this part. With traditional approaches we need to describe the object textually, which is a difficult task. A more intuitive way to perform a query is presenting the object itself or some kind of visual representation as a key to access the database, rendering a textual description unnecessary.

This paper describes some of the key components of a visual object retrieval system which can be used for finding objects in a database by presenting a sample view or a sequence of sample views.

Object recognition systems usually have the disadvantage that they require training images of the objects. With increasing size of the object set, this becomes more and more tedious, since objects usually have to be presented manually. We aim to eliminate this process by using CAD data for the training process.

This paper is organized as follows. Section 2 mentions a subset of related systems. Section 3 contains a short description of the Axiom system's architecture,

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 253–267, 2002.

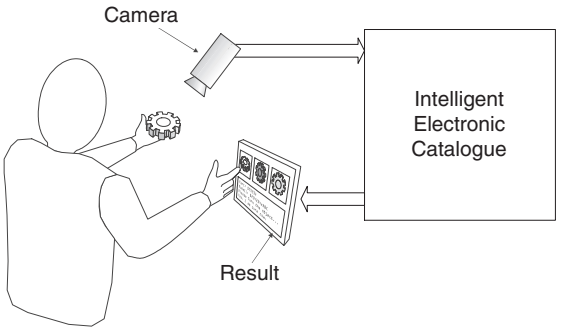


Fig. 1. The application framework for AXIOM

Sect. 4 discusses some implementation issues. In Sect. 5, we present some preliminary experimental results. In Sect. 6, we draw conclusions from the results presented here and describe our plans for future work.

2 Related Work

The system we describe here stems from a combination of two widely spread topics of research: Object Recognition (OR) and Visual Information Retrieval (VIR). Both fields try to attain the goal of making a computer system recognize objects or images the same way humans do. A summary of important topics is presented in the next paragraphs.

2.1 Visual Information Retrieval

The appearance of multimedia data has spawned a tremendous amount of interest on retrieval of non-textual data of any kind, including visual data like images or video data, and aural data like speech, music, or sound samples. Visual Information Retrieval is mostly related to finding images in a database based on the content or a graphical description of a query image. The principle is to compute features of the images in the database which can be accessed via an efficient index. There are many research systems, some of which have evolved into commercial applications, such as Virage [1] and QBIC [2]. However, none of them addresses the retrieval of 3D-objects; they are focused on finding similar images, not necessarily similar objects. The extracted features usually lack rotation, scaling and shift invariances, which are indispensable in 3D object recognition systems.

2.2 Object Recognition

One can distinguish between model-based and view-based object recognition. Model-based recognition tries to match a three-dimensional model with a sample

view of the object. In contrast, view-based object recognition directly uses sample views and tries to obtain a correlation with known views labeled with the object identification. A widely used approach is to extract features from the sample views which are invariant against in-plane transformations and robust against distortions like partial occlusion. The resulting feature vectors are then classified using statistical or artificial intelligence algorithms. The result of this process is either the identification of the recognized object or a list of possible objects.

For a comprehensive survey of older systems, see [3]. Two of the most interesting recent systems are SEEMORE [4] and DyPERS [5]. SEEMORE uses a combination of color, texture and shape descriptions for view-based object recognition. Classification is performed using a nearest-neighbor based approach. MIT's DyPERS can cope with the presentation of an arbitrary view of an object by using multidimensional receptive field histograms for statistical object recognition and localization. A system which combines approaches from object recognition and VIR is Nefertiti [6], an access engine for content-based retrieval of images and three-dimensional models.

AXON^2 [7,8,9] is a flexible system which is able to combine different features and classification methods.[1] Our system is the successor of AXON^2 and provides a larger variety of image processing and classification algorithms.

3 System Description

3.1 Global Structure

The system is organized as a collection of building blocks which can be combined to form a visual object retrieval engine. In most cases, the structure is similar to the one shown in Fig. 2.

The image acquisition module captures an image from either a camera, a file, the database or a network connection. The system supports a variety of cameras, ranging from a simple web cam to high-end industrial cameras. The acquired image may undergo a processing stage which performs normalization and object-background segmentation.

Alternatively, the system can generate images by rendering CAD data. This can be achieved by modeling the camera and lighting properties, rendering the requested images, and feeding them into the system just like camera images. The rendering process can be performed automatically once the modeling has been done and the surface material data have been converted into a representation usable for rendering engines. This is useful especially for training the system, since the tedious task of presenting objects and acquiring images manually is avoided.

[1] An interactive web demo can be found at
`http://www.techinfo.rwth-aachen.de/Forschung/VIR/Axon/demo/`

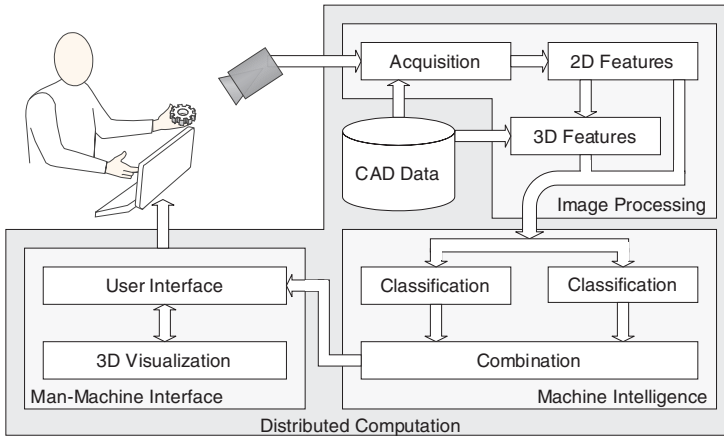


Fig. 2. The structure of the object recognition system

A subsequent module extracts feature vectors from the processed images. The system contains various modules for different kinds of features, describing color, texture, or shape.[2]

The feature vectors are then assigned to the corresponding object by a single classifier, an ensemble of classifiers or a hierarchy of classifiers. Finally, the result is presented to the user.

3.2 Image Acquisition and Processing

The first stage of any visual recognition process is always the acquisition of images. In AXIOM they can be captured using different color cameras. Depending on the objects to be recognized, the quality of a simple commercial web cam may be sufficient, but most of the time more expensive industrial cameras delivering images of higher quality are inevitable.

For object sets including both dark and bright elements, it may be necessary to continuously adapt the camera parameters, in order to optimally use its dynamic range. This simulates the adaption process of the human visual system.

In order to provide a better image intensity adaption than those provided by modern cameras, we have developed an algorithm based on generalized linear discriminants [10], which is used to learn how to change the camera shutter speed in order to make optimal use of the camera's dynamic range. To decide when a shutter value is acceptable, an intensity descriptor for an image is computed, which must lie within a defined interval.

Two shutter speed estimators run in parallel (Fig. 3). A deterministic one will try to adapt the shutter with the smallest possible step for the given camera. Its

[2] In this article, we omit descriptions of shape features because they are not used in the experiments described later.

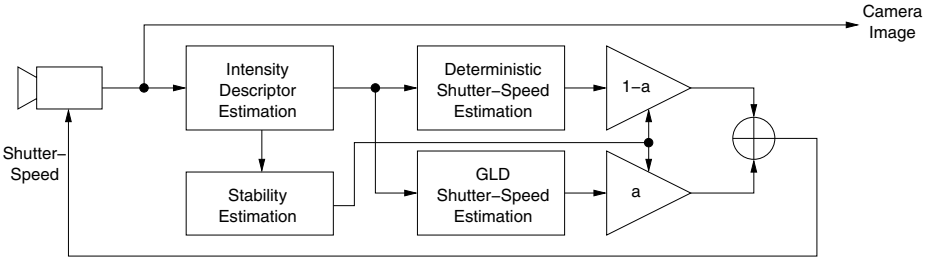


Fig. 3. Camera shutter-speed adaption

estimations are rather conservative, but it will ensure a stable image intensity and the convergence of the intensity descriptor into the desired interval. The main disadvantage are the required long adaption times. The second estimator uses the results of the first one to learn how to adapt the shutter in a more efficient way. A reliability factor is adapted that controls a weighted result formed from the outputs of both estimators. When the image intensity becomes unstable, the first estimator is considered as more reliable.

3.3 Segmentation

For working with global features which describe an entire image, all objects in the image need to be detected and separated from the background. The high complexity of this task, known as *segmentation*, relies mainly on the fact that the detection cannot be achieved without high-level knowledge on the appearance and distribution of the objects in the captured scene. In [11] a general approach is introduced that allows consideration of high level cues into the segmentation using Bayesian Belief Networks. The main idea here is to split the image into a set of small regions, such that each object contains at least one region and the borders between objects and background are also borders between the found regions. Using the positions and colors of background and expected objects, and other similar high-level cues, probability maps are generated that allow to find regions that have a high probability of belonging to one of the objects to be detected. These regions are then used as seeds in a region growing algorithm that uses similarity measures between regions to decide if they need to be merged or not.

3.4 Feature Extraction

From the segmented image some feature vectors are extracted that describe the contents of the image in a more compact form.

A three-dimensional object recognition system like Axiom requires feature vectors to be invariant against translation and rotation (similarity transformations on the image plane) and against small changes in the illuminant. Partial

occlusion can also be expected. For these reasons, only a subset of color, texture and form features can be utilized.

Each object detected in the segmentation can be analyzed as a whole (global features) or just salient parts of it can be considered locally for further analysis (local features). Global features usually have the disadvantage that they require a separation of the object from the background and are more susceptible to distortions and occlusions. Therefore, care must be taken to avoid these shortcomings. In contrast, local features do not require a segmentation, but they cannot describe the coarse structure of an object. The selection of relevant or conspicuous regions can be achieved using saliency analysis (e. g. [12]) or by explicitly selecting the locations using approaches similar to the one introduced in [13].

Here, a set of "locations" is selected, each one specified by its position in the image, a radius and an angle. Not only information about color, form or texture of each local region can be considered, but also their geometrical configuration.

The C++ code for algorithms described here has been released under the GNU Lesser General Public License (LGPL) in
`http://ltilib.sourceforge.net/`

Color Constancy. Before extracting color features, it can be mandatory to normalize the image colors in order to eliminate the effect of changes in the illumination. Finlayson et. al. [14] have shown that, given some conditions, a diagonal transformation suffices for color constancy.

For a color channel $c(x, y)$ and a norm $\|\cdot\|$, it can be easily shown that the normalization of each color channel $R(x, y)$, $G(x, y)$ and $B(x, y)$ of the RGB color space with their respective norms yields a canonical image which is invariant against changes in the illuminant color. In [14] the norm $\|c(x, y)\| = \mu(c(x, y))/3$ is used. $\mu(c(x, y))$ represents the mean value of all pixels in channel $c(x, y)$. We prefer the norm $\|c(x, y)\| = \mu(c(x, y)) + \kappa\sigma(c(x, y))$ with κ constant and $\sigma(c(x, y))$ the standard deviation of all elements of $c(x, y)$. This norm keeps the majority of the new canonical channel values in the original value range. This solves the problem that, using the the original transform, many new canonical values can be out of range depending on the distribution of the original values. In our experiments, the value $\kappa = 3$ was used. This way, if we assume a normal distribution, the probability that the value lies within the original range, is 99.7%.

Chromaticity Histograms. Since the introduction of color histogramming by Swain and Ballard [15], this principle has been applied and enhanced in many object and image recognition systems, e. g. SEEMORE [4].

The chromaticity histogram contains exclusively information about color, ignoring the intensity, which is usually encoded in the channels $r = \frac{R}{R+G+B}$ and $g = \frac{G}{R+G+B}$. This fact is used in [14] to produce canonical images invariant against changes of the lighting geometry, which are reflected exclusively in the intensity channel. To increase the robustness against noise, the resulting 2D

histograms are low-pass filtered. All ideally black pixels are ignored; this way only the object pixels are considered.

The normalization and low-pass filtering of the histogram yields an approximation for the color probability distribution by a mixture of Gaussians, where the filter kernel corresponds to the basis function of the mixture model [10].

Color Histograms. The chromaticity feature discards valuable information for the recognition. Considering the previous generation of a canonical image invariant against changes in the illuminant color, the intensity information can also be robustly evaluated using simple color histograms, generated in the following way. Four 1D histograms are calculated for the red, green, blue and luminance channels, where luminance is defined as $L = \frac{min(R,G,B)+max(R,G,B)}{2}$. Again, to reduce the effects of noise a 1×3 low-pass filter kernel is applied to each histogram. The four histograms are concatenated into a 128-dimensional vector.

Multiresolutional Energy Feature. Using the concepts for texture channels presented in [16], a texture-energy image is generated. Smith's suggestions for scale and rotation invariance were also considered. The algorithm used is similar to the generation of a conspicuity map for the intensity channel in [12] except that faster QMF wavelets are used instead of the more time-consuming Gabor filter banks.

A channel is first transformed into its wavelet representation (with four resolutions). The low-pass channel is ignored, which means that only the three following bands are considered (nine channels in total). The *energy* channels are computed applying $|\cdot|^2$ to each pixel of each spectral channel. They are then combined into a single image using the multiscaled addition defined in [12].

These energy images are calculated for the three color channels R, G and B. Histograms with 32 bins for each channel are concatenated into a 96 dimensional feature vector. The same measures against noise were taken as for the color features.

Multiresolutional Oriented Gaussian Derivative Feature. The information contained at each specific spectral band of a multiresolutional image decomposition has been successfully applied in image retrieval tasks (for example [17]). These features, however, have a limited applicability in 3D object recognition due to their lack of invariance against scaling and rotation. Rotation invariant and scaling equivariant features can be generated exploiting the steerability property of oriented Gaussian derivatives (OGD). We described in [18] that the energy of an oriented channel extracted with a n-th order gaussian derivative filter is given by:

$$E_R^\theta = A_0 + \sum_{i=1}^{n} A_i \cos(2i\theta - \Theta_i) \tag{1}$$

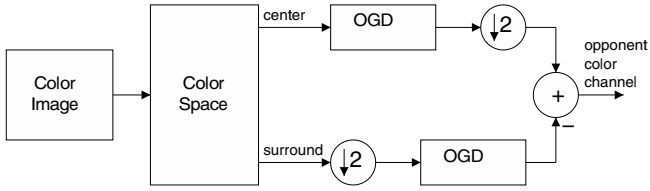


Fig. 4. Generation of an opponent color channel

where the A_i do not depend on the steering angle θ, i. e. the *descriptors* A_i are rotation invariant.

The original channel can be analyzed extracting the descriptors A_i and Θ_i at different resolutions using for example dyadic pyramid architectures. This way, different frequency bands can be considered in the final feature vector. We employ a dyadic pyramid of OGDs with standard deviation $\sigma_m = \sigma_0 2^m$ for each level m used.

There is also a local variant of these features: The A_i and Θ_i terms are computed for just one resolution, depending on the radius of the location used.

Multiresolutional Opponent Color Feature. Until now, the color information has been considered using the R, G and B color channels separately. We have also developed another approach for color analysis which is inspired by the opponent color theory of human vision. Here, receptive fields organized in a center-surround fashion compare the stimulus of the center (e. g. red) with the stimulus in the surrounding (e. g. green). This can be modeled using the scheme shown in Fig. 4.

If the energy E^{θ}_{a-b} for the opponent color channel is calculated as described in [18], it follows:

$$E^{\theta}_{a-b} = E^{\theta}_{a} + E^{\theta}_{b} - E^{\theta}_{a \leftrightarrow b} \tag{2}$$

Comparing this term with the previous features, it is clear that the new information gained comparing both center and surround channels is contained in the *opponent color energy* $E^{\theta}_{a \leftrightarrow b}$. For the OGD, this term can be also expressed as

$$E^{\theta}_{a \leftrightarrow b} = A^{ab}_{0} + \sum_{i=1}^{n} A^{ab}_{i} \cos(2i\theta - \Theta_i) \tag{3}$$

Combining the descriptors A_0 and A_1 of both input channels (center R and surround G) and the new opponent color channel, a feature vector with 48 dimensions is generated.

3.5 Classification

There are several types of classification algorithms which have proven useful for pattern recognition. Among them are neural networks, maximum-likelihood classifiers, and variants of k-nearest neighbor algorithms. Each of them is suitable for

a different kind of classification problem. For interpretability and the possibility of building hybrid classifier ensembles, all of them are designed to model the a-posteriori distribution of the object label given the image (or feature) data.

k Nearest Neighbors. The method of k nearest neighbors [19] is very common and works well for many cases. It is easy to implement and can be cast into a traditional database system. The idea of k nearest neighbor is to store all training vectors in an appropriate data structure. A query consists in finding the k training vectors which lie closest to the query vector. The resulting class label is then computed from the class labels of the training vectors found. In a probabilistic framework, an appriately designed KNN classifier may be interpreted as a maximum likelihood classifier using mixtures of univariate gaussians as data models, where each data point is the center of a gaussian. Using this classifier requires all feature vectors to be stored in a data structure which allows efficient indexing, such as a kd-tree [20].

Neural Networks. We have integrated several kinds of neural networks. The major advantage of neural network classifiers is that they require no feature vectors to be retained. The object label is computed instead of searched in the database. In out experiments, the most successful type has been a Radial Basis Function network [10], which is a standard architecture consisting of two fully connected layers; the first one computing the response of a univariate gaussian to the input pattern, and the second one performing a linear transform of the output vector of the first layer.

The second weight matrix is trained by applying the Moore-Penrose pseudo-inverse. The position and variance of the first layer gaussians is determined by an approach devised in [21], using Linear Vector Quantization (LVQ) for the position of the prototypes and heuristics for their standard deviations. We have slightly modified this method and use a different variant of LVQ known as OLVQ3 [22]. For determining the variance, we use the minimum distance between two points of different classes. This generates more tight prototypes than the original approach of using the distance of the prototypes, thus improving classification when discrimination among classes is hard.

Maximum-Likelihood Classifiers. Maximum-Likelihood classifiers consist of a collection of data models, one for each class of objects.

It is well known that selecting the data model with the highest a-posteriori probability given an input vector is guaranteed to yield the lowest possible bayes error, provided that each data model actually represents the true data distribution of its class. If we assume equal a-priori probabilities for all classes, and a uniform a-priori distribution of the data, this is equivalent to selecting the model with the highest likelihood of having generated the input vector. Thus, the difficulty is to find correct data models. Common data models we have implemented include multivariate gaussian models and sparse histograms.

Maximum likelihood classifiers have a property which makes them superior for applications with changing data sets. Whereas the vast majority of neural network architectures cannot be extended to recognize new objects, all that is required for extending a maximum-likelihood classifier is adding a new data model.

3.6 Postprocessing

Classification Assessment. As a means to enhance classification performance, the system can record statistics of the actual performance of a classifier. We evaluate the classification results on a validation set and compute the probability $P(C|o,x)$ for the real class being C when the classifier selects class o. For simplicity, we assume C being conditionally independent of x given o. Then, the distribution for the classes can be calculated by $P(C|x) = \sum_{o \in K} P(C|o) \cdot P(o|x)$ where K is the set of all classes.

Ensembles. As another means of improving classification performance, we have integrated several methods to combine classifiers to ensembles. The most effective method is to perform weighted averaging of the individual results. Then, $P(C|x) = \sum_{k \in K} P(C|k,x) \cdot P(k|x)$ with $P(k|x)$ being the probability that classifier k returns the correct result for input x. Often, it is sufficient to assume $P(k|x) = P(k)$, with $P(k)$ being the a priori probability of correct classifications of classifier k.

Sequence Combination. The combination of results taken from image sequences are performed as follows: A sequence is taken as a n-dimensional vector of independent variables $(x_1, \ldots, x_n)$. Then, the classification result is obtained by $P(C|x_1, \ldots, x_n) = \prod_{x_i} P(C|x_i)$. Even though the images of a sequence are not really independent, neglecting these dependencies yields sufficient results as well.

4 Implementation Issues

The AXIOM system needs to cope with different kinds of requirements that have to be addressed in an actual implementation in order to handle complex tasks.

- Flexibility: The system must not implement a fixed methodology but must allow for the experimental parts of object recognition research.
- Modularity: It must be easy to implement and use new functionality.
- Programmability: The system must be programmable so that fixed algorithms can be stored, executed and tested with different parameters.
- Parallelizability: Since we use complex algorithms and have to handle large amounts of data, we need to make use of all available computing hardware.

Obviously, these issues are unrelated to computer vision or artificial intelligence, but rather involve software design; nonetheless they are important for the success of a practical AI system. For accomplishing the goals mentioned above, we chose to implement a variant of a dataflow machine which works across machine boundaries.

The global system is implemented in Java2, whereas time-consuming individual modules are implemented in C++ using the LTIlib.[3] It provides over 150 modules and contains over 1000 classes.

4.1 Dataflow

For the Axiom system, we developed our own programming language which can be used to express functional programs as well as dataflow programs; the Forcs language (Functional Object Retrieval Control Syntax).

Besides a subset of the Scheme language, it contains several special functions for controlling Axiom and calling its functional blocks. Its syntax is derived from Scheme. With Forcs, it is possible to create functional programs as well as dataflow graphs [23].

The advantage of the dataflow principle is that the parallelism which is inherent in the algorithm is expressed in a natural way. Thus it is not necessary to pre-determine the statements that are executed in parallel, since a parallel execution schedule can easily be determined automatically.

4.2 Distributed Implementation

In order to handle the computational load imposed by the various image processing and classification modules, the system is able to perform distributed computing on several computers. In order to enable a cooperation of heterogenous computer systems, the distribution mechanism relies on standardized protocols and runtime environments. We chose Java2 as implementation language and CORBA 2.3 as distribution mechanism.

In our implementation, Axiom can run as a server that offers methods for evaluating dataflow graphs via CORBA. For instance, an Axiom system running the user interface and possibly the camera interface, can contact servers in the network and distribute the computational load among the available servers. An important question with this mechanism is how to select the server for a given task. Since the available schedulers for CORBA are not sufficient for our requirements, we developed a dynamic scheduler which assigns each task to the machine which fits the task's requirements best.

5 Experimental Results

The applicability of visual object retrieval as an intelligent tool for accessing 3D-databases obviously depends on its recognition correctness. However, for in-

[3] Available from `http://ltilib.sourceforge.net/`

Table 1. Recognition rates for three different test object sets

object set	views	1-best	2-best	3-best
COIL-100	3600	100%		
42 plush obj.	840	98.8%	99.9%	100%
202 plush obj.	7272	87.9%	93.8%	95.8%

teractive applications it is not generally necessary to perform an unique identification, but rather to find the most *similar* objects. Therefore, it is sufficient if the system presents the most similar objects at the top of the result list. In order to incorporate the position of the desired object in this ranking list, we use the concept of k-best recognition, which is basically a function mapping the rank k to the rate of the desired object being at rank k or higher in the result list. In the experiments described below, we used only global features because they showed a higher recognition rate in preliminary tests than local features.

5.1 Real Training Images

The system has been tested on three different object sets with real training images. The first one is the COIL database [24], consisting of 100 objects, each one represented by 72 views. 36 of these views were used for training, the remaining were used for testing. The second set is composed of views of 42 plush animals, most of them articulated. Training was performed using 35 random views for each object, the test set consisted of of 840 novel random views, 20 for each object. Both training and test images were taken from objects presented manually by an operator, thus showing partially occluded objects. Lighting conditions were the same for training and test images. The third set contains views of 202 plush animals, some of them were articulated. 40 views, taken from camera points uniformly distributed on an upper hemisphere above the object, served as training set. The test set consisted of 36 novel views for each object.[4] For the test images, the objects were presented manually, resulting in partial occlusion caused by the operator's hand (Fig. 5 right). Lighting conditions were the same for the training and test set. For classification we used an ensemble of five RBF networks classifying global features, namely chromaticity histogram, color histogram, opponent color feature, wavelet energy feature, and oriented gaussian derivative feature.

Table 1 shows the k-best recognition rates for these three sets. The COIL database is an easily solvable problem. The 42 plush objects are slightly harder to recognize because of more similarities among some of them. This effect is even stronger with the larger object set of 202 objects. Some of the objects look completely alike, such that it seems probable that human test subjects would reach a classification performance similar to the one of the system.

[4] This data set is exhibited on the WWW at
`http://www.techinfo.rwth-aachen.de/Forschung/VIR/Axon/demo`

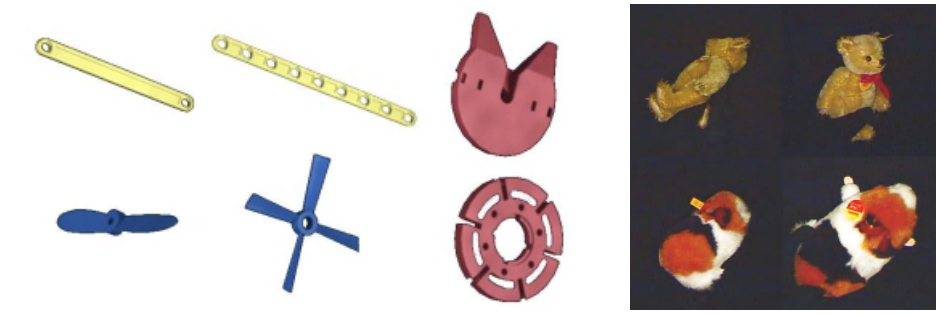


Fig. 5. Left side: Rendered training images for six objects from the "fischertechnik" set (background omitted). Right side: Test images of four of the 202 objects from the plush animal set

Table 2. Recognition rates for three tests with the "fischertechnik" object set. "sequence" is the number of images per sequence, "n" denotes the number of tests per object

sequence	n	1-best	2-best	3-best
1	40	62.7%	79.1%	86.7%
3	13	68.0%	83.7%	89.6%
5	8	76.1%	86.4%	93.1%

5.2 Training from CAD Data

In order to demonstrate the feasibility of using computer-generated images for training, we used an object set composed of 47 objects taken from the modeling kit "fischertechnik" (examples are shown on the left side of Fig. 5). We modeled the properties of our image acquisition hardware with a raytracer and generated 216 training images of each object. For the test images, we used camera images of the real objects. Thus, training and test images have different lighting conditions. Furthermore, some details were not contained in the rendered images (e.g. different surface properties on an object). We exlusively used a maximum likelihood classifier with the multiresolutional opponent color feature described in Sect. 3.4 because, in preliminary experiments, these features outperformed all other feature types, most notably shape descriptors. Table 2 shows recognition rates for single test images as well as for image sequences of various lengths.

5.3 Evaluation

In our experiments we have found that even in the presence of partial occlusion, global features showed superior performance than local features. The plush animal dataset could be processed successfully using exclusively color and texture features. The objects are usually deformed by the operator's hand, which can

cause test images to show shapes completely different from the ones contained in the training images of the same object.

The "fischertechnik" set shows an interesting property of our multiresolutional OGD feature. Even though it was designed as a texture feature, it seems also to be a descriptor for the shape of an object, suggesting that it is applicable to a wide variety of object recognition tasks.

6 Conclusions and Future Work

We have presented a visual object retrieval system which is flexible enough to handle different kinds of object retrieval and image processing tasks.

We have demonstrated that it is possible to achieve reasonable recognition results by using exclusively global color and texture features. The latter ones can also be used as shape descriptors.

For objects with rigid shape, it seems a feasible approach to use computer-generated images for training. The recognition results are promising, but require further improvements. We plan to achieve this by incorporating features describing three-dimensional object properties.

The system has demonstrated its applicability for an object set of medium size. For large object sets of thousands of different objects, we are currently extending and improving the AXIOM system in order to handle object sets large enough for real-world applications. Local feature classification can be used to improve the system's recognition accuracy. Even though they have failed in our experiments to provide good results as the sole information source, we are currently working on using them as a means to discriminate objects of the same coarse shape by detecting details. This leads to an architecture roughly similar to the one found in the human brain, where wholes are recognized in a different area than details [25]. Finally, we are working on more complex classifiers that are able to include spatial and temporal relationships within image sequences.

Acknowledgements. This project has been funded by the Heinz-Nixdorf Foundation. The test objects were kindly provided by Margarete Steiff GmbH and fischerwerke Artur Fischer GmbH & Co. KG.

References

1. Gupta, A., Jain, R.: Visual Information Retrieval. Communications of the ACM **40** (1997) 71–79
2. Flickner, M., Sawhney, H., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D., Yanker, P.: Query by image and video content: The QBIC system. IEEE Computer **28** (1995) 23–32
3. Pope, A.R.: Model-based object recognition – a survey of recent research. Technical report 94–04, University of British Columbia (1994)
4. Mel, B.W.: SEEMORE: Combining Color, Shape, and Texture Histogramming in a Neurally Inspired Approach to Visual Object Recognition. Neural Computation **9** (1997) 777–804

5. Schiele, B., Pentland, A.: Probabilistic object recognition and localization. In: Proc. Int. Conf. on Computer Vision – ICCV'99. (1999) 177–182
6. Paquet, E., Rioux, M.: Nefertiti: a query by content system for three-dimensional model and image databases management. Image and Vision Computing **17** (1999) 157–166
7. Elsen, I.: Ansichtenbasierte 3D-Objekterkennung mit erweiterten Selbstorganisierenden Merkmalskarten. VDI-Verlag (2000)
8. Elsen, I., Kraiss, K.F., Krumbiegel, D., Walter, P., Wickel, J.: Visual Information Retrieval for 3D Product Identification. K nstliche Intelligenz **1/99** (1999) 64–67
9. Walter, P.: Verfahren der sequentiellen Merkmalsanalyse für die Mustererkennung. Shaker Verlag (2000)
10. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press (1995)
11. Alvarado, P., Berner, A., Akyol, S.: Combination of high-level cues in unsupervised single image segmentation using bayesian belief networks. In: Proc. of the Int. Conf. on Imaging Science, Systems and Technology, Las Vegas, Nevada (2002)
12. Itti, L., Koch, C., Niebur, E.: A model of saliency-based visual attention for rapid scene analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence **20** (1998) 1254–1259 Saliency maps based on multiresolution analysis.
13. Lowe, D.G.: Object recognition from local scale-invariant features. In: International Conference on Computer Vision, Corfu, Greece (1999) 1150–1157
14. Finlayson, G.D., Schiele, B., Crowley, J.L.: Comprehensive colour image normalization. In: ECCV. (1998)
15. Swain, M., Ballard, D.H.: Color indexing. International Journal of Computer Vision **7** (1991) 11–32
16. Smith, J.R.: Integrated Spatial and Feature Image Systems: Retrieval, Analysis and Compression. PhD thesis, Columbia University (1997)
17. Jain, A., Healey, G.: A multiscale representation including opponent color features for texture recognition. IEEE Transaction on Image Processing **7** (1998) 124–128
18. Alvarado, P., Dörfler, P., Wickel, J.: Axon2 – a visual object recognition system for non-rigid objects. In: Proc. of the IASTED Int. Conf. Signal Processing, Pattern Recognition and Applications, Rhodes, Greece (2001) 235–240
19. Duda, R.O., Hart, P.E.: Pattern Classification and Scene Aanalysis. John Wiley & Sons (1973)
20. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM **18** (1975) 509–517
21. Vogt, M.: Combination of Radial Basis Function Neural Networks with Optimized Learning Vector Quantization. In: ICNN-93, IEEE Int. Conf. on Neural Networks. Volume 3. (1993) 1841–1846
22. Kohonen, T.: Self-Organizing Maps. Springer-Verlag (1995)
23. Wickel, J.: The Forcs language. Internal report, Lehrstuhl für Technische Informatik, RWTH Aachen (2002)
24. Nene, S.A., Nayar, S.K., Murase, H.: Columbia Object Image Library (COIL-100). Technical Report CUCS-006-96, Columbia University (1996)
25. Carter, R.: Mapping the Mind. University of California Press (1999)

Representation of Behavioral Knowledge for Planning and Plan-Recognition in a Cognitive Vision System

Michael Arens and Hans-Hellmut Nagel

Institut für Algorithmen und Kognitive Systeme
Fakultät für Informatik der Universität Karlsruhe (TH)
Postfach 6980, D–76128 Karlsruhe, Germany
{arens,nagel}@ira.uka.de

Abstract. The algorithmic generation of textual descriptions of image sequences requires conceptual knowledge. In our case, a stationary camera recorded image sequences of road traffic scenes. The necessary conceptual knowledge has been provided in the form of a so-called *Situation Graph Tree (SGT)*. Other endeavors such as the generation of a synthetic image sequence from a textual description or the transformation of machine vision results for use in a driver assistance system could profit from the exploitation of the same conceptual knowledge, but more in a planning (*pre*-scriptive) rather than a *de*-scriptive context.
A recently discussed planning formalism, *Hierarchical Task Networks (HTNs)*, exhibits a number of formal similarities with SGTs. These suggest to investigate whether and to which extent SGTs may be re-cast as HTNs in order to re-use the conceptual knowledge about the behavior of vehicles in road traffic scenes for planning purposes.

1 Introduction

Road traffic offers challenging examples for an algorithmic approach towards 'understanding' image sequences of such scenes. We are concerned here with systems which rely on an *explicated* knowledge base which comprises knowledge about the geometry of the depicted scene and about admissible trajectories of moving vehicles. In addition, conceptual knowledge is required in order to transform quantitative results related to geometric properties into textual descriptions of vehicle maneuvers and their context – see, e. g., [7,9]. In these cases, the required knowledge has been provided in the form of a so-called *Situation Graph Tree (SGT)*.

An attempt to invert such a processs, i. e. to generate a synthetic image sequence from a textual description related to the same discourse domain of road traffic ([17]), access to the same or similar *conceptual* knowledge turns out to be desirable in order to infer details which have not been incorporated into the text because they were assumed to be available to a human in the form of commonsense knowledge. Driver assistance systems based on video cameras

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 268–282, 2002.

installed in vehicles and linked to machine vision systems constitute another example where conceptual knowledge about road traffic scenes will be needed, in particular if such assistance systems have to cope with the intricacies of innercity traffic. Preliminary results in this direction ([5,6]) suggest the exploitation of knowledge originally provided for the interpretation of image sequences. In both cases, the conceptual knowledge may be needed for planning purposes. This consideration motivates our investigation to convert knowledge provided in the form of an SGT into another form more suitable for planning. Due to a number of formal similarities, *Hierarchical Task Networks (HTNs)* offer themselves as a good starting point for such an investigation.

In the sequel, we first sketch the formulation of planning tasks based on HTNs and then outline SGTs. Based on this exposition, we discuss our experience with a more detailed comparison between these two formalisms. As will be seen, a number of aspects which become apparent only upon closer inspection requires careful attention: the context in which these two formalisms have been developed influenced their properties.

2 Hierarchical Task Networks

Planning with Hierarchical Task-Networks (HTNs) [21] (see also [20]) is similar to STRIPS-like planning formalisms [4]. Each world-state (the state of the discourse in which a plan is searched) is represented as a set of atoms true in that state. An action modifying the world-state in STRIPS normally corresponds to a triple of sets of atoms: the preconditions, which have to be true whenever the action should be executed, the add-list of atoms, listing all atoms which will become true after the action has been executed, and the delete-list holding all atoms negated (or deleted) from the world-state by executing the action. A planning-problem in STRIPS can then be stated as a tuple $\mathbf{P} = \langle \mathcal{I}, \mathcal{G}, A \rangle$, where $\mathcal{I}$ denotes the initial world-state, $\mathcal{G}$ describes the desired world-state and A is the set of possible actions. A solution to such a planning problem is given by a sequence of actions, which – starting with the initial world-state – produce the desired world-state. Each atom contained in the desired world-state is called a *goal*.

In HTN-Planning, the goals of STRIPS are replaced by *goal-tasks*. In addition to these tasks, two other types of tasks can occur: *primitive tasks* and *compound tasks* [1]. While primitive tasks can be accomplished by simply executing an associated action, compound-tasks are mapped to one or more possible *task networks*. These networks are directed graphs comprising further tasks as nodes and successor relations between these tasks as edges, which naturally define the order in which tasks should be achieved. In addition to that, the edges can be attributed by further constraints concerning the preconditions of tasks and the assignment of variables comprised in task-descriptions. If a task network consists of only primitive tasks, it is called *primitive task network*.

Planning with HTNs can thus be formulated as follows [2]: The planning domain is described as a pair $\mathcal{D} = \langle A, M \rangle$, where A again denotes the set of

possible actions and M is a set of mappings from compound tasks onto task networks. A planning problem can then be formulated as a tuple $\mathbf{P} = \langle d, \mathcal{I}, \mathcal{D} \rangle$ where d is an initial task network, $\mathcal{I}$ describes the initial world-state and $\mathcal{D}$ is a planning domain. Note that – in contrast to STRIPS – no desired world-state ($\mathcal{G}$) is given in the formulation of the planning problem. A solution to the problem $\mathbf{P}$ can thus not simply be a sequence of actions achieving such a world-state. Instead, each goal which has to be achieved by the plan is incorporated into the initial task network d as a goal task. The solution to $\mathbf{P}$ in HTN planning is a task network itself: given the initial task network d of $\mathbf{P}$, an expansion of that network is searched such that the resulting network is primitive and all constraints incorporated into the network are satisfied. If d is a primitive task network, then d itself is a solution to the problem $\mathbf{P}$ if all constraints in d are satisfied. Otherwise d contains at least one compound task c. For this task a mapping $m \in M$ is searched which maps (expands) the task c onto a further task network d_c. This means that one way to achieve the task c is to accomplish all the tasks in the network d_c. The result is a task network d' similar to d, but incorporating the new network d_c instead of the one task c. If the resulting network d' is primitive, a solution is found, otherwise another compound task can be expanded into a network and so on.

[3] have shown that HTN-Planning is strictly more expressive than STRIPS-style planning. [11] points out that – speaking about efficiency of planning formalisms – HTN-Planning profits from the fact that the user can *direct* the search for plans by defining the mapping from compound tasks to task networks and thus restrict the search space to those decompositions that lead to allowed or desired plans. While in STRIPS-style planners any sequence of actions leading from the initial state to the desired goal state is a valid plan, in HTN-planning only those sequences that can be derived by decomposing the initial task network into a primitive task network are supposed to be meaningful and feasible. By defining this mapping from compound tasks into task networks, the user not only directs the search for a problem solution, but also incorporates more knowledge about the planning domain into the planner.

2.1 Formal Syntax for HTNs

In the following formal definition of HTN-syntax we use the notation introduced by [13], where o_i denotes a primitive task and N_i stands for a non-primitive (or compound) task. The *mappings* from compound tasks onto task networks of [2] are called *reduction schemes* in [13]. Similar to [13] we will write $r_{i,j} = \langle \mathbf{C} \rangle$, which denotes that $r_{i,j}$ is the jth reduction scheme of the non-primitive task N_i, where $\mathbf{C}$ is a set of constraints describing a task network. These reduction schemes define the decomposition of non-primitive tasks into task networks: reduction schemes are defined as task networks using the 14 types of constraints shown in Fig. 1. After formulating a hierachical task network in terms of these 14 constraints, [13] define transformation schemes based on [12] which *encode* the defined network in form of propositional logic formulae. The solution to the initial planning problem is then found by the so-called *planning as satisfiability*-

(1)	o_i:<action_name>	The primitive task symbol o_i is mapped onto the action name <action_name>.
(2)	N_j:<task_name>	The non-primitive task N_j is mapped onto the task name <task_name>.
(3)	$o_p \prec o_q$	The primitive task o_p temporally precedes the primitive task o_q.
(4)	$o_s \prec N_p$	The primitive task o_s temporally precedes the non-primitive task N_p.
(5)	$N_p \prec o_s$	The non-primitive task N_p temporally precedes the primitive task o_s.
(6)	$N_p \prec N_q$	The non-primitive task N_p temporally precedes the non-primitive task N_q.
(7)	$o_p \xrightarrow{f} o_q$	There exists a causal link between o_p and o_q such that f is an add effect of o_p and a precondition of o_q.
(8)	$o_p \xrightarrow{f} N_q$	There exists a causal link between o_p and N_q such that f is an add effect of o_p and a precondition of N_q.
(9)	$N_q \xrightarrow{f} o_p$	There exists a causal link between N_q and o_p such that f is an add effect of N_q and a precondition of o_p.
(10)	$N_p \xrightarrow{f} N_q$	There exists a causal link between N_p and N_q such that f is an add effect of N_p and a precondition of N_q.
(11)	$o_q \xrightarrow{f} ?$	The primitive task o_q has an add effect which can be used to supply a precondition of some other task not known a-priori.
(12)	$N_p \xrightarrow{f} ?$	The non-primitive task N_p has an add effect f which can be used to supply a precondition of some other task not known a-priori.
(13)	$? \xrightarrow{f} o_p$	The primitive task o_p has a precondition f which has to be supplied by some other task not known a-priori.
(14)	$? \xrightarrow{f} N_q$	The non-primitive task N_q has a precondition f which has to be supplied by some other task not known a-priori.

Fig. 1. The 14 constraints used by [13] to express task networks. (1) and (2) state a syntactical mapping. The remaining twelve constraint types result from the fact that two relations dominate the structure of a single task network: the two-digit temporal order relation (defined on two kinds of arguments, primitive and non-primitive tasks) leads to four constraint types ((3)–(6)). The also two-digit causal link relation is based on three different types of arguments: primitive and non-primitive tasks, and a *wild card* for not-specified tasks. As the relation $? \xrightarrow{f} ?$ would make no sense here, this leads to the remaining eight types of constraints ((7)–(14)).

paradigm: Any model of the formulated formulae will correspond to a plan solving the planning problem.

Other approaches to HTN-planning use a similar formulation of the HTN-structure itself, but the planning on that structure is done by an algorithm which tentatively (and recursively) decomposes tasks and backtracks, if constraints are violated or no solution for the planning problem can be found with the actually decomposed task-network. Examples for this approach are SHOP (see [14]) and SHOP2 (see [15]), but also UMCP (see [1]).

3 Situation Graph Trees

As outlined in Sect. 1, we use Situation Graph Trees (SGTs) for supplying behavioral knowledge to our vision system. In this formalism, the behavior of an agent is described in terms of situations an agent can be in. Transitions between these situations express a temporal change from one situation to another. The term *generically describable situation* [16] or *situation scheme* describes the situation of an agent schematically in two parts: a *state scheme* denotes the state of an agent and of his environment with logic predicates. If each of these *state atoms* is satisfied, the agent is *instantiating* this situation scheme. The other part of a situation scheme is called *action scheme*. This scheme denotes actions – again formulated in logic predicates – the agent is supposed or expected to execute if this situation scheme can be instantiated.

Due to the analysis of image sequences, a natural discretization of time is given by the temporal interval between two consecutive image frames. For each of these frames, a quantitative evaluation of the image is performed, which leads to a *quantitative description* of the depicted scene. The results of this evaluation are associated with concepts formulated in logic predicates, yielding a *conceptual description* for each point in time (each image frame) of the observed scene.

An observed agent in the scene should instantiate one situation described in an SGT for each point in time. The *expectation* of what situation the agent will instantiate at the next point in time can be expressed by so called *prediction edges*. A prediction edge always connects two situation schemes, meaning that if an agent has been recognized to instantiate the situation from which the edge starts, one probable[1] next situation for that agent could be the situation to which the edge points. Of course, an agent can persist in a single situation for more than one point of time. Thus a *self-prediction* – a prediction edge from a situation scheme to itself – has to be allowed in SGTs.

Situation schemes together with prediction edges build *situation graphs*. These graphs are directed – according to the definition of prediction edges – and can comprise cycles. Situation schemes inside such a graph can be marked as *start*

[1] Note that SGTs and the SGT-traversal described later in this section are *deterministic* formalisms, in contrast to *probabilistic* Bayesian-Networks (see [19]) utilized for example by [10] and [23]. Thus prediction edges deterministically define which (and in which order) situation schemes should be investigated.

situation and/or *end situation*. Each path from a start situation to an end situation defines a *sequence* of situations represented by the situation graph. To refine a single situation scheme, it has to be connected to one or more situation graphs by so called *specialization edges*. Imagine for example an image sequence of an innercity road intersection with multiple lanes for each direction. On the most abstract level of conceptual description it might be sufficient to describe an agent (some vehicle in the scene) to just cross the intersection (see Fig. 2). To instantiate such a situation scheme `cross_0`, an observed object (`Agent`) would only have to satisfy two predicates: `agent(Agent)`, stating that the actual object is indeed an agent of some action, and `traj_active(Agent)`, meaning that the actual object is the object of interest (read: the *trajectory* of that object is actually *active*). The only action atom of this situation scheme in Fig. 2 is to print a message whenever this scheme is instantiated (`note(cross(Agent))`). This description might then hold during the whole period of time this vehicle is depicted in the image sequence (and recognized by the vision system). For a more detailed description of what the observed agent is doing, one might divide the whole process of crossing an intersection into one situation in which the agent advances towards the intersection on a certain lane (`drive_to_intersection_1`), another one, in which the agent actually crosses the intersection area (`drive_on_intersection`), and a third one, which describes the agent as leaving the intersection on another lane (`leave_intersection_1`, see Fig. 2). These three situation schemes are temporally ordered in the way they are mentioned above, so they should be connected with prediction edges according to that order, marking the first one as a start- and the last one as an end situation. Each of these situation schemes might hold for an agent for more than one image frame, so they should also be connected to themselves by self-predictions. By connecting the more general situation scheme describing an agent as crossing an intersection with the specializing situation graph constructed above, one states that everytime an agent can be described as crossing an intersection, he might also be in one of the situations comprised in that graph. The situation scheme specialized in this way is called *parent situation scheme* of the specializing situation graph.

Of course, several more specializations of the situation scheme describing an agent as crossing an intersection are imaginable: the agent might turn left or right, he might follow or precede another vehicle, etc. . Multiple specializations of one situation scheme are possible in SGTs simply by connecting that parent scheme with several other situation graphs. Analogously, every situation scheme comprised in such a specializing graph might be specialized further. In the example above, while driving to the intersection, the agent might have to stop (`stop_in_front_of_intersection_2`), wait (`wait_in_front_of_intersection_2`), and start again (`start_in_front_of_intersection_2`), or he might proceed without stopping (`proceed_to_intersection_2`).

However – for reasons explained later in this section – it is not allowed to connect situation schemes and situation graphs in a way that cycles from a more general situation scheme via some more special situation schemes back to again more general situations exist in the SGT. For the same reasons one situation

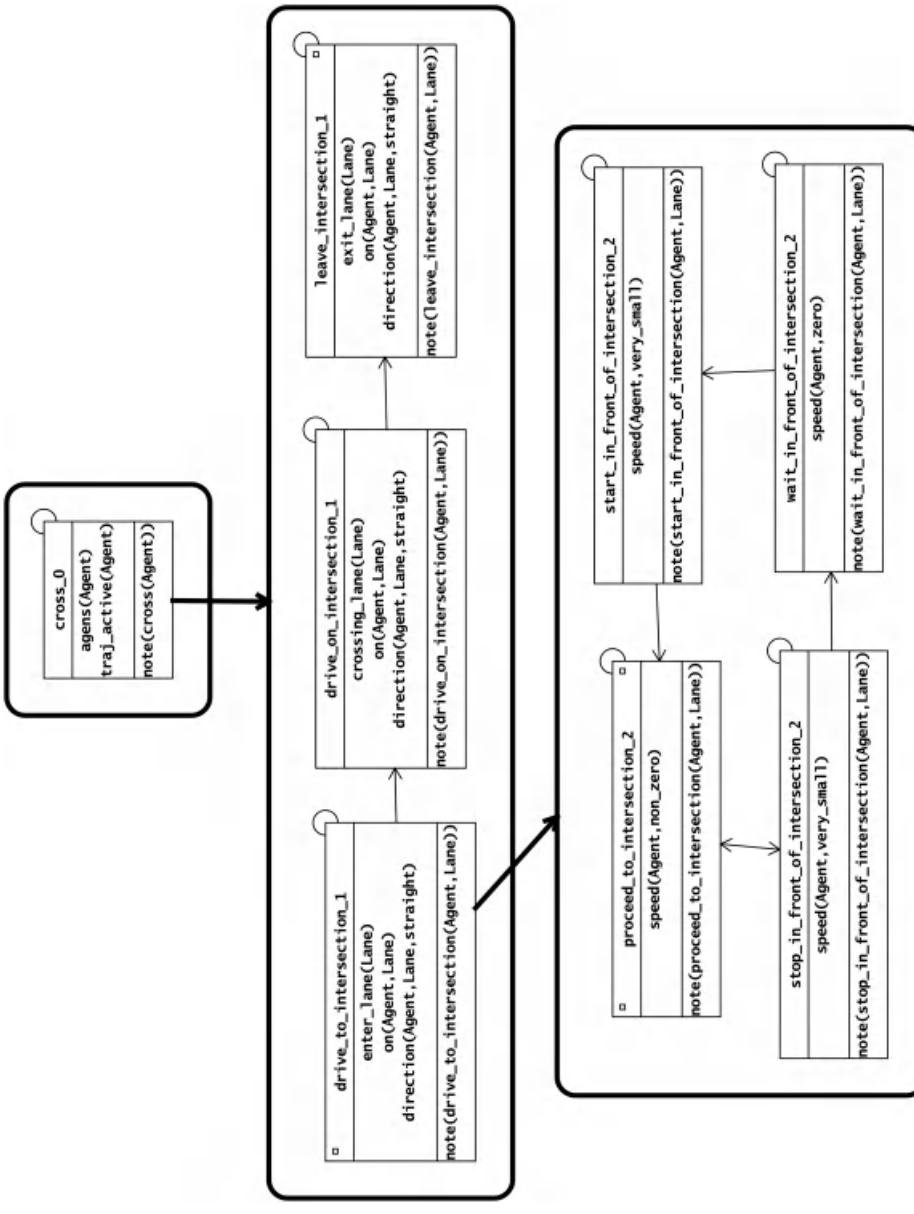


Fig. 2. Part of the situation graph tree describing the behavior of vehicles on an inner city road intersection taken from [18]. Rounded rectangles depict situation graphs. Situation schemes are shown as normal rectangles. Thin arrows stand for prediction edges, while bold arrows represent specialization edges. Circles in the right upper corner of situation schemes indicate a self-prediction of that scheme. Small rectangles to the left or to the right respectively of the name of situation schemes mark that scheme as a start- or end-situation. Detailed description in the text.

graph can only specialize exactly one or none situation scheme. Thus, situation schemes connected recursively to situation graphs build a tree-like structure, a situation graph tree.

The recognition of situations an agent is instantiating for each point in time could obviously be done by simply testing each and every situation scheme comprised in the SGT. Fortunately, this time-consuming approach can be significantly sped up by utilizing the knowledge encoded in form of prediction edges and specialization edges. The recognition of situations is thus performed by a so called *graph traversal*: The recognition of situations – and thus the traversal of the SGT – is started in the root situation graph[2]. Here, a start situation is searched for which each state atom is satisfied for the actual inspected point of time. If no such situation scheme can be found, the traversal fails. Otherwise the observed agent is instantiating that situation. If this situation scheme is specialized further by any situation graph, these graphs are again searched for start situation schemes with state atoms also satisfied in the actual point of time. If such a situation scheme can be found, it is instantiated by the agent. In this way the most special situation scheme that can be instantiated by the agent is searched. For the next point in time, only those situation schemes are investigated, which are connected by prediction edges to the situation scheme instantiated last. This also means that only situation schemes are investigated, which are members of the same graph. Thus, a prediction for a situation scheme on the same level of detail is searched. If none such scheme can be instantiated, two cases have to be distinguished: if the situation scheme last instantiated is an end situation, the failed prediction on this level of detail is accepted and the traversal is continued in the parent situation[3] of the actual graph. Otherwise (i.e., no end-situation was reached in the actual graph), the traversal fails completely. Thus, for each point of time, an agent is instantiating situation schemes on several levels of detail, each on a path from a scheme in the root graph to the most special scheme.

Concerning the action schemes comprised in these situation schemes, again two cases have to be distinguished. Each action atom in an action scheme can be marked as *incremental* or *non-incremental*. Incremental actions of a situation scheme are executed whenever this situation scheme lies on a path of schemes currently instantiated by the agent. Non-incremental actions of a situation scheme are only executed whenever this situation scheme is the most special scheme on a path of schemes an agent is instantiating.

[2] Here we suppose that an SGT comprises exactly one root situation graph. If a constructed SGT comprises more than one situation graph without a parent situation scheme, one single root graph can be introduced consisting of only one abstract situation scheme. This situation scheme can then be specialized by all situation graphs previously without a parent situation scheme.

[3] According to the definition of allowed specialization edges, the parent situation scheme of a situation graph is always unique for all graphs except the root graph of the SGT. In case of a failing prediction inside the root graph, the traversal fails completely, because no prediction is feasible inside the graph and no more general description of the behaviour of an agent exists.

3.1 Formal Syntax for SGTs

For a formal declaration of SGTs, [22] developed the declarative language `SIT++`. In this language, all aspects of SGTs described above can be expressed. The traversal of such an SGT is done by executing a logic-program obtained by an automatic translation of the `SIT++`-formulated SGT into rules of a *fuzzy metric temporal Horn logic* (FMTHL), also developed by [22]. By translating the SGT-traversal into a set of *fuzzy* logic rules, it is possible to operate with fuzzy truth values, which are essential to cope with the *uncertainty* arising in image evaluation due to sensor noise, but also with the inherent *vagueness* of natural language concepts. By translating the SGT-traversal into a set of *metric temporal* logic rules, it is possible to explicitly reason about time and thus for example demand minimum and maximum durations for the instantiation of particular situation schemes.

4 Comparison of HTNs and SGTs

Both – HTNs and SGTs – describe states and transitions between such states for a certain discourse in a hierarchical manner. In HTN-planning – as in every other planning task – a sequence of atomic actions is searched which solves a given problem. Thus, the structure of HTNs is action-oriented. In contrast, the graph traversal on SGTs used in our vision system *describes* situations and sequences of situations an observed agent is instantiating. Therefore the SGT-structure is state-oriented. It appears natural, though, to identify the state-atoms of a situation scheme in SGTs with preconditions of a task in HTNs (compare Fig. 1, (13) and (14)), as in both formalisms these atoms (state atoms and preconditions, respectively) have to be satisfied in order to execute the associated action(s). The action atoms of a situation scheme should then be identified with actions in HTNs. Because situation schemes can comprise more than one action atom, there cannot exist a one-to-one correspondence between situation schemes and (primitive) tasks. In case of more than one action atom inside a situation scheme, this scheme can only be identified with a non-primitive task (compare Fig. 1, (2)), which then should be decomposed into a sequence of primitive tasks mapped onto actions (compare Fig. 1, (1)) each corresponding to one action atom of the initial situation scheme.

The add- and delete-effects of an action are explicitly stated in HTNs as causal links between tasks (compare Fig. 1, (7)–(12)). In SGTs, only the association of states and corresponding actions are explicitly incorporated into a situation scheme. The effects of a certain action are not modelled inside the SGT, but are obtained from sensors, namely the underlying image sequence evaluation system[4]. Thus to facilitate planning with SGTs, either the add- and

[4] The need for a representation of *fuzzy* measures as well as for an explicite representation of time and a metric on time is a direct consequence of the connection of symbolic knowledge inside SGTs and the sensor evaluation outside the SGT.

delete-effects of actions have to be modelled inside the SGT or the sensor input has to be simulated[5].

The prediction edges of SGTs can be compared to the temporal relations included in the definition of task networks of HTNs (compare Fig. 1 (3)–(6)), though they cannot be totally identified: the temporal order-relation on tasks is (as an order-relation) *transitive*. The prediction edges of situation graphs define a relation on situation schemes which is not necessarily transitive (and normally is not transitive at all).

HTNs in the presented formalism do not naturally support loops inside a task network. Especially the analogy to self-predictions of situation schemes in SGTs can only be formulated in the HTN-syntax by introducing an abstract non-primitive task N_i, which then is decomposable into (at least) two other task networks: one (reduction scheme $r_{i,0}$) consisting of the final non-primitive task ($N_{i'}$) and another one ($r_{i,1}$), which consists of that final task ($N_{i'}$) and the initial task N_i again (additionally a temporal order between these tasks should be asserted, e.g., by the statement $N_{i'} \prec N_i$). In this way, an iterative execution of the task $N_{i'}$ is expressed by means of recursive reduction of non-primitive tasks, where $r_{i,0}$ represents the recursion end (compare, e.g., [11]).

The reduction schemes in the HTN-formalism correspond to the specialization edges leading from general situation schemes to situation graphs in SGTs, though there exists a difference in the utilization of these hierarchy-generating elements in both structures: in SGTs, all specialization edges and the situation graphs they point at are included in the SGT right from the start. This *entire* SGT is traversed then in order to find situation schemes which an observed agent instantiates. HTN-planning in contrast is started with one single initial task network (describing the planning problem). This task network is iteratively expanded to a *hierarchical* task network by decomposing non-primitive tasks. This means that one single SGT incorporates the complete knowledge about the (admissable) behaviour of agents in an observed discourse, whereas one single HTN only represents a single plan in the discourse for which a plan is searched. Thus, an SGT should not be compared to a single HTN, but rather to all HTNs which can be build with one set of primitive tasks, non-primitive tasks, and reduction schemes.

4.1 Expressing SGT-Knowledge in the HTN-Formalism

In the following, we assume that effects of actions of an SGT are obtained from a simulation as described in [8]. The simulation used there was controlled by

[5] This approach of sensor input simulation has already been applied by [8]: In case of an occlusion in the observed scene, sometimes not enough visual information can be obtained by the image evaluation system to properly actualize the estimated (quantitative) state of an observed agent. In such a case, [8] replaced the actualization of the state of the observed agent by a simulated motion model, controlled by action atoms obtained from SGT-traversal. This is done until enough information can be obtained from consecutive images again, e.g., due to an dissolving occlusion in the scene.

an incremental recognition of situations. The name of the actually instantiated situation scheme was printed to a stream. This stream was analysed in each (temporal) step, which led to new state atoms for the next point of time according to the simulation model. The only action needed here therefore is a print-command. To express the knowledge represented by the SGT depicted in Fig. 2, we start with an initial task-network d. This network contains only one non-primitive task N_0, which is bound to the task-name `pre_cross_0(Agent)`. As explained in the previous section, this pre-task is needed to express the self-prediction of the situation scheme `cross_0`. The preconditions of this task are given by the state atoms of the situation scheme `cross_0`. Thus, the complete initial task network can be formulated as

$$d = \left\langle \begin{array}{c} N_0 : \texttt{pre\_cross\_0}(\texttt{Agent}), \\ ?\ \overset{\texttt{agent(Agent)}}{\longrightarrow} N_0\ ,\ ?\ \overset{\texttt{traj\_active(Agent)}}{\longrightarrow} N_0 \end{array} \right\rangle .$$

To express the self-prediction of the scheme `cross_0`, we use two reduction schemes for the non-primitive task N_0, namely

$$r_{0,0} = \left\langle \begin{array}{c} N_1 : \texttt{note}(\texttt{cross}(\texttt{Agent})), \\ N_0 : \texttt{pre\_cross\_0}(\texttt{Agent}), \\ N_1 \prec N_0 \end{array} \right\rangle$$

and

$$r_{0,1} = \left\langle\, N_1 : \texttt{note}(\texttt{cross}(\texttt{Agent}))\, \right\rangle .$$

The scheme `cross_0` was specialized by a graph consisting of three consecutive situation schemes. Each of these schemes again was connected to itself by a self-prediction. Thus, we get the following reduction scheme for the non-primitive task N_1:

$$r_{1,0} = \left\langle \begin{array}{c} N_2 : \texttt{pre\_drive\_to\_intersection\_1}(\texttt{Agent}), \\ N_3 : \texttt{pre\_drive\_on\_intersection\_1}(\texttt{Agent}), \\ N_4 : \texttt{pre\_leave\_intersection\_1}(\texttt{Agent}), \\ ?\ \overset{\texttt{enter\_lane(Lane)}}{\longrightarrow} N_2\ ,\ ?\ \overset{\texttt{on(Agent,Lane)}}{\longrightarrow} N_2, \\ ?\ \overset{\texttt{direction(Agent,Lane,straight)}}{\longrightarrow} N_2, \\ ?\ \overset{\texttt{crossing\_lane(Lane)}}{\longrightarrow} N_3\ ,\ ?\ \overset{\texttt{on(Agent,Lane)}}{\longrightarrow} N_3, \\ ?\ \overset{\texttt{direction(Agent,Lane,straight)}}{\longrightarrow} N_3, \\ ?\ \overset{\texttt{exit\_lane(Lane)}}{\longrightarrow} N_4\ ,\ ?\ \overset{\texttt{on(Agent,Lane)}}{\longrightarrow} N_4, \\ ?\ \overset{\texttt{direction(Agent,Lane,straight)}}{\longrightarrow} N_4, \\ N_2 \prec N_3, N_3 \prec N_4 \end{array} \right\rangle .$$

Again, each of the three pre-tasks (N_2, N_3, and N_4) has to be supplied with two reduction schemes expressing the self-prediction of the situation schemes to which they correspond. We thus obtain the two reduction schemes

$$r_{2,0} = \left\langle \begin{array}{c} N_5 : \texttt{note}(\texttt{drive\_to\_intersection}(\texttt{Agent}, \texttt{Lane})), \\ N_2 : \texttt{pre\_drive\_to\_intersection\_1}(\texttt{Agent}), \\ N_5 \prec N_2 \end{array} \right\rangle ,$$

$$r_{2,1} = \left\langle N_5 : \texttt{note}(\texttt{drive\_to\_intersection}(\texttt{Agent}, \texttt{Lane})) \right\rangle,$$

and two similar reduction schemes for each of the tasks N_3 and N_4. The transformation of the specializing situation graph of the scheme `drive_to_intersection_1` into the HTN-formalism is somewhat more complicated: This graph contains several loops in addition to the self-predictions, each starting and ending in the situation scheme `proceed_to_intersection`, because this is the only start- and end-situation of that graph. A transformation of this graph can only be done by translating each path from a start-situation to an end-situation into a separate reduction scheme. As the graph to be transformed here also contains self-prediction, the transformation of paths from start- to end-situation has to be done first. This leads to the following reduction schemes:

$$r_{5,0} = \left\langle \begin{array}{c} N_8 : \texttt{pre\_pre\_proceed\_to\_intersection\_2}(\texttt{Agent}), \\ ? \xrightarrow{\texttt{speed(Agent,non\_zero)}} N_8 \end{array} \right\rangle.$$

denotes the path starting in `proceed_to_intersection_2(Agent)` and ending there, visiting no other situation scheme. Another path visiting the situation scheme `stop_in_front_of_intersection_2(Agent)` is represented by

$$r_{5,1} = \left\langle \begin{array}{c} N_8 : \texttt{pre\_pre\_proceed\_to\_intersection\_2}(\texttt{Agent}), \\ N_9 : \texttt{pre\_pre\_stop\_in\_front\_of\_intersection\_2}(\texttt{Agent}), \\ N_{10} : \texttt{pre\_pre\_proceed\_to\_intersection\_2}(\texttt{Agent}), \\ ? \xrightarrow{\texttt{speed(Agent,non\_zero)}} N_8 \ , \ ? \xrightarrow{\texttt{speed(Agent,very\_small)}} N_9, \\ ? \xrightarrow{\texttt{speed(Agent,non\_zero)}} N_{10}, \\ N_8 \prec N_9, N_9 \prec N_{10} \end{array} \right\rangle.$$

The last path finally visits all four situation schemes of the graph and is represented by

$$r_{5,2} = \left\langle \begin{array}{c} N_8 : \texttt{pre\_pre\_proceed\_to\_intersection\_2}(\texttt{Agent}), \\ N_9 : \texttt{pre\_pre\_stop\_in\_front\_of\_intersection\_2}(\texttt{Agent}), \\ N_{11} : \texttt{pre\_pre\_wait\_in\_front\_of\_intersection\_2}(\texttt{Agent}), \\ N_{12} : \texttt{pre\_pre\_start\_in\_front\_of\_intersection\_2}(\texttt{Agent}), \\ N_{10} : \texttt{pre\_pre\_proceed\_to\_intersection\_2}(\texttt{Agent}), \\ ? \xrightarrow{\texttt{speed(Agent,non\_zero)}} N_8 \ , \ ? \xrightarrow{\texttt{speed(Agent,very\_small)}} N_9, \\ ? \xrightarrow{\texttt{speed(Agent,zero)}} N_{11} \ , \ ? \xrightarrow{\texttt{speed(Agent,very\_small)}} N_{12}, \\ ? \xrightarrow{\texttt{speed(Agent,non\_zero)}} N_{10}, \\ N_8 \prec N_9, N_9 \prec N_{11}, N_{11} \prec N_{12}, N_{12} \prec N_{10} \end{array} \right\rangle.$$

The pre-pre-tasks introduced in the reduction schemes above can then be further reduced with respect to self-prediction as exemplified above. Thus, the complete SGT described in Sect. 3 can be translated into the HTN-formalism of [13].

5 Conclusion

In the preceding section, SGTs and HTNs have been compared. A single SGT incorporates the complete knowledge about the behavior of agents in a discourse, while a single HTN always expresses a single plan in the world modelled by tasks and reduction schemes. Thus an SGT is rather comparable to all HTNs that can be build with these tasks and reduction schemes than to a single HTN. In other words: *One HTN is comparable to one instance of the schematic behavior description given by an SGT.* As a further result it can be stated that SGTs and HTNs correspond structurally in most aspects except for the following details:

- The (partial) temporal ordering of tasks in HTNs is an order-relation, while the relation defined on situation schemes by prediction edges is more general.
- Because of the strict temporal order between tasks, HTNs do not naturally support loops, though loops and self-predictions of SGTs can also be modelled with HTNs.
- HTNs explicitly denote add- and delete-effects of tasks. In SGTs, the effects of actions are obtained by sensor input from an observed (simulated) world.

Due to the considerable structural correspondences between SGTs and HTNs, the difference between the planning task conducted on the HTN-structure and the observation task utilizing SGTs predominantly lies in the algorithm performing the planning or the observation task respectively. One way to facilitate a planning task with the knowledge encoded in an SGT could therefore be the translation of the SGT into an HTN-syntax as outlined in Sect. 4. One of the HTN-planning algorithms mentioned in Sect. 2.1 could then be applied to the resulting set of tasks and reduction schemes. Because SGTs do not denote effects of action atoms, these effects would either have to be incorporated into the resulting HTN-formulation, or they would have to be obtained by simulated sensor input as described in [8]. This way of planning on SGTs would *adapt* the underlying knowledge-structure and leave the algorithm performed on that structure *as is*.

Another way to facilitate planning using the knowledge given in the form of an SGT is to modify one of the HTN-planning algorithms mentioned in Sect. 2.1 in a way that it can cope with the representation of SGTs and the differences with respect to the HTN-formulation arising due to that representation. Again, the results of action atoms comprised in the SGT would either have to be incorporated into the SGT or obtained from a simulation. The result of such an approach would be an *adapted* planning-algorithm, running on a knowledge-structure that could be left *as is*.

An analogous conclusion can be drawn concerning the feasibility of the HTN-formalism for plan-recognition or observation-tasks: if SGT- and HTN-formalisms are as far matchable as implied by the comparison drawn in Sect. 4, one could think of using the HTN-formulated knowledge to recognize (or observe) the behavior of an agent in the discourse domain for which the HTN was designed. As stated above regarding the utilization of SGT for planning, this could again be done by either a transformation of the HTN-formulated knowledge into an

SGT, on which the graph-traversal outlined in Sect. 3 would then be applied, or the algorithm performing the graph-traversal could be adapted to the HTN-formalism.

6 Future Work

The correspondencies found between the SGT- and HTN-formalisms suggest a possible automatic transformation of one formalism into the other and vice versa, though further investigations have to show if such a transformation is algorithmically feasible. The possible use of the knowledge coded in HTNs for an *observation task* is interesting, but not our primary goal due to our experience with SGTs. We rather want to try to implement an HTN-planning algorithm on SGTs, *adapt* them to that *planning task* as much as neccessary without losing their capabilities for the observation task. The resulting structure could then (hopefully) be used for both, planning and observation tasks.

References

1. K. Erol, J. Hendler and D. S. Nau: *UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning.* In: K. J. Hammond (Ed.): Proc. of the 2nd Int. Conf. on Artificial Intelligence Planning Systems (AIPS-94), June 13–15, 1994, University of Chicago, Chicago, Illinois, 1994, pp. 249–254.
2. K. Erol, J. Hendler and D. S. Nau: *HTN Planning: Complexity and Expressivity.* Proc. of the 12th National Conf. on Artificial Intelligence (AAAI–1994), Volume 2. Seattle, Washington, USA, July 31–August 4, 1994. AAAI Press, 1994, pp. 1123–1128.
3. K. Erol, J. Hendler and D. S. Nau: *Complexity Results for HTN Planning.* Annals of Mathematics and Artificial Intelligence **18**:1 (1996) 69–93.
4. R. E. Fikes and N. J. Nilsson: *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.* Artificial Intelligence **2** (1971) 189–208.
5. K. Fleischer and H.-H. Nagel: *Machine-Vision-Based Detection and Tracking of Stationary Infrastructural Objects Beside Innercity Roads.* In: Proc. of the IEEE Intelligent Transportation Systems Conf. (ITSC), August 25–29, 2001, Oakland, CA, USA, pp. 525–530.
6. K. Fleischer, H.-H. Nagel, and T. M. Rath: *3D-Model-Based-Vision for Innercity Driving Scenes.* In: Proc. of the IEEE Intelligent Vehicle Symposium (IV-2002) June 18–20, 2002, Versailles, France.
7. R. Gerber: *Natürlichsprachliche Beschreibung von Straßenverkehrsszenen durch Bildfolgenauswertung.* Dissertation, Fakultät für Informatik der Universität Karlsruhe (TH), Karlsruhe, Januar 2000; erschienen im elektronischen Volltextarchiv der Universität Karlsruhe, `http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=2000/informatik/8` (in German).
8. M. Haag: *Bildfolgenauswertung zur Erkennung der Absichten von Straßenverkehrsteilnehmern.* Dissertation, Fakultät für Informatik der Universität Karlsruhe (TH), Karlsruhe, Juli 1998; erschienen in der Reihe Dissertationen zur Künstlichen Intelligenz (DISKI) **193**; infix-Verlag Sankt Augustin 1998 (in German).

9. M. Haag and H.-H. Nagel: *Incremental Recognition of Traffic Situations from Video Image Sequences.* Image and Vision Computing **18**:2 (2000) 137–153.
10. R. J. Howarth and H. Buxton: *Conceptual Descriptions from Monitoring and Watching Image Sequences.* Image and Vision Computing **18**:2 (2000) 105–136.
11. S. Kambhampati: *A Comparative analysis of Partial Order Planning and Task Reduction Planning* SIGART Bulletin **6**:1 (1995) 16–25.
12. H. Kautz, D. McAllester, and B. Selman: *Encoding Plans in Propositional Logic.* In: L. C. Aiello, J. Doyle, and S. C. Shapiro (Eds.): Proc. of the 5th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, 1996; Morgan Kaufman, San Mateo, CA, USA 1996, pp. 374–384.
13. A. D. Mali and S. Kambhampati: *Encoding HTN Planning in Propositional Logic.* In: R. G. Simmons, M. M. Veloso, and S. Smith (Eds.): Proc. of the 4th Int. Conf. on Artifical Intelligence Planning Systems (AIPS-98), Pittburgh, Pennsylvania, USA, 1998; AAAI-Press, 1998, pp. 190–198.
14. D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila: *SHOP: Simple Hierarchical Order Planner.* In: Th. Dean (Ed.): Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99), Stockholm, Sweden, July 31 – August 6, 1999; Morgan Kaufmann, San Mateo, CA, USA 1999, pp. 968–973.
15. D. S. Nau, H. Muñoz-Avila, Y. Cao, A. Lotem, and S. Mitchell: *Total-Order Planning with Partially Ordered Subtasks.* In: B. Nebel (Ed.): Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI-2001), Seattle, Washington, USA, August 4–10, 2001; Morgan Kaufman, San Mateo, CA, USA 2001, pp. 425–430.
16. H.-H. Nagel: *From Image Sequences towards Conceptual Descriptions.* Image and Vision Computing **6**:2 (1988) 59–74.
17. H.-H. Nagel, M. Haag, V. Jeyakumar, and A. Mukerjee: *Visualization of Conceptual Descriptions Derived from Image Sequences.* Mustererkennung 1999, 21. DAGM-Symposium, Bonn, 15.–17. September 1999, Springer-Verlag, Berlin, Heidelberg, u.a. 1999, pp. 364–371.
18. H.-H. Nagel: *Natural Language Description of Image Sequences as a Form of Knowledge Representation.* In: W. Burgard, T. Christaller, and A. B Cremers (Eds.): Proc. of the 23rd Annual German Conf. on Artificial Intelligence (KI-99), Bonn, Germany, September 13–15, 1999, LNCS **1701**, Springer-Verlag, Berlin, Heidelberg, u.a. 1999, pp. 45–60.
19. J. Pearl: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufman, San Mateo, CA, USA 1988.
20. S. Russel and P. Norvig: *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, New Jersey, USA, 1995.
21. E. D. Sacerdoti: *Planning in a Hierarchy of Abstraction Spaces.* Artificial Intelligence **5** (1974) 115–135.
22. K. H. Schäfer: *Unscharfe zeitlogische Modellierung von Situationen und Handlungen in der Bildfolgenauswertung und Robotik.* Dissertation, Fakultät für Informatik der Universität Karlsruhe (TH), Karlsruhe, Juli 1996; erschienen in der Reihe Dissertationen zur Künstlichen Intelligenz (DISKI) **135**; infix-Verlag Sankt Augustin 1996 (in German).
23. G. Socher, G. Sagerer, and P. Perona: *Bayesian reasoning on qualitative descriptions from images and speech.* Image and Vision Computing **18**:2 (2000) 155–172.

Qualitative Velocity and Ball Interception

Frieder Stolzenburg*, Oliver Obst, and Jan Murray**

Universität Koblenz-Landau, AI research group
Universitätsstr. 1, D–56070 Koblenz, GERMANY
{stolzen,fruit,murray}@uni-koblenz.de

Abstract. In many approaches for qualitative spatial reasoning, navigation of an agent in a more or less static environment is considered (e.g. in the double-cross calculus [12]). However, in general, real environment are dynamic, which means that both the agent itself and also other objects and agents in the environment may move. Thus, in order to perform spatial reasoning, not only (qualitative) distance and orientation information is needed (as e.g. in [1]), but also information about (relative) velocity of objects (see e.g. [2]). Therefore, we will introduce concepts for qualitative and relative velocity: (quick) to left, neutral, (quick) to right. We investigate the usefulness of this approach in a case study, namely ball interception of simulated soccer agents in the RoboCup [10]. We compare a numerical approach where the interception point is computed exactly, a strategy based on reinforcement learning, a method with qualitative velocities developed in this paper, and the naïve method where the agent simply goes directly to the actual ball position.

Keywords: cognitive robotics; multiagent systems; spatial reasoning.

1 Motivation

There is some literature on qualitative spatial reasoning where the subject of consideration is motion observation and representation (see e.g. [7]). But only few references can be found dealing with objects moving in the environment of the subject. It also seems so far that velocity of objects is not considered in the qualitative reasoning community. However, in robotics it is necessary to deal with velocity of both the robot and objects in the environment. Taking robotic soccer in the RoboCup as an example, we have to reason about velocities of the ball, team mates and adversaries.

1.1 Robotic Soccer as Example Scenario

In order to evaluate different approaches for dealing with velocity and compare them with each other, we make use of the RoboCup soccer simulator. The RoboCup [10] is an international research and education initiative, attempting to foster artificial intelligence and intelligent robotics research by providing a standard problem where a wide range of

* New address: Frieder Stolzenburg, Hochschule Harz, Friedrichstr. 57–59, 38855 Wernigerode, Germany, fstolzenburg@hs-harz.de.

** The authors are partially supported by the grants *Fu 263/6-1* and *Fu 263/8-1* from the German research foundation *DFG*.

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 283–298, 2002.

technologies can be integrated and examined. There are annual tournaments in different leagues with real robots of different sizes or virtual, i.e. simulated robots. The authors of this paper are involved in the simulation league (see e.g. [6]) with simulated soccer agents using the Soccer Server [3], a physical soccer simulation system.

The Soccer Server is a system that enables autonomous agents consisting of programs written in various programming languages to play a match of soccer (association football) against each other. A match is carried out in a client/server style: a server provides a virtual field and simulates all movements of a ball and players; each client controls movements of one player; communication between the server and each client is done via UDP/IP sockets. The Soccer Server consists of two programs, the server itself and a monitor. The server program simulates the ball and players movements, communicates with clients, and controls a game according to the rules. All games are visualized by displaying the field of the simulator by the soccer monitor on a computer screen.

In general, reasoning about velocities can be of interest in dynamic environments to estimate when and if at all one agent can reach another agent or moving object. In the case of physical soccer robots, accurate formulæ for object movements may not be available or difficult to obtain. The RoboCup simulation league, where these formulæ are known to the robots, seems to be an ideal environment to test whether qualitative processing of velocity is possible at all and how a qualitative computation performs in contrast to a numeric one.

1.2 Overview of the Paper

In the sequel, we first introduce several quantitative methods devoted to solve our example problem, namely ball interception (Sect. 2). After this, we discuss qualitative approaches from the literature and introduce a new method that makes use of qualitative direction and velocity information (Sect. 3). Then, we evaluate all methods introduced in this paper, showing that the qualitative method does not perform too badly in contrast to the other approaches (Sect. 4). Finally, we state conclusions (Sect. 5).

2 Ball Interception with Numerical Methods

Information about velocity is especially important in applications, where spatial agents are situated in a highly dynamic environment. This means, not only the agent moves, but also objects or other agents in the environment move and change their position, possibly with high or varying speed. In the following, we consider robotic soccer as application scenario, which has the desired properties. In particular, we will address the problem of ball interception. Thus, we have the situation that the soccer ball is rolling with a certain velocity towards a certain direction. Now it is the task of the soccer agent to reach the ball as fast as possible.

2.1 Soccer Simulation in the RoboCup

The optimal method for ball interception obviously is to compute the interception point exactly, considering relative position, orientation and velocity of the agent and the ball,

and then let the agent go to this point as quickly as possible. We must take into account the physical laws for the ball and player movement. This means, that the ball becomes slower and slower, the longer it rolls, and every player certainly has a maximal speed for running. Regarding all this might lead to a fairly complicated procedure for computing the interception point, depending on how precise and realistic the modeling of physical laws such as friction etc. is done.

Fortunately, the ball movement model and also the player dash model that is implemented in the Soccer Server is relatively simple, so that by a straightforward iterative procedure, the interception point can be computed effectively. We will develop the respective formulæ further down. Unfortunately, in this model, it is assumed that the ball acceleration $\vec{a}$ – that is the derivative of its velocity $\vec{v}$ wrt. the time t – is negatively proportional to $\vec{v}$, i.e. $\vec{a} \sim -\vec{v}$. However, this does not correspond to any standard physical phenomena such as friction, where $\vec{a}$ is constant, or air resistance, where we have $\vec{a} \sim -\vec{v}^{\,2}$.

In consequence, this means that the derivation for computing the interception point (in Sect. 2.2) is only valid in the context of the movement model in the Soccer Server. This is certainly one reason, why we should address the problem with a qualitative approach (see Sect. 3.3). What are in general the advantages of a qualitative approach?

1. Qualitative approaches are likely to be more robust wrt. changes in the world model, because they abstract or approximate the physical reality, which is far more complicated and unreliable than the Soccer Server world. Each change in the world model requires an adaption of a hard-coded quantitative method.
2. It is an interesting question on its own to compare the performance of qualitative and quantitative approaches. Even if the exact computation of the interception point can be done (as for the Soccer Server see below), a qualitative method may be simpler and not worse than a quantitative method.
3. Another motivation to try it with a qualitative approach is that it is likely that real (human) soccer players do not solve differential equations while chasing the ball, but use a cognitively more plausible method. But note that the Soccer Server is an artificial scenario: ball and player velocity approximately are in the same order of magnitude, whereas in real soccer the ball is much faster than the player. A similar observation holds for animals catching their prey.

2.2 Computing the Interception Point

Let us now derive the formulæ for computing the exact interception point. It is used in our reference method for the comparison with other approaches. In the Soccer Server, the ball speed at time t is calculated as $v(t) = \mu \cdot v(t-1)$ with $\mu < 1$. Therefore, if the velocity of the ball is v_0 at time $t = 0$, we have $v(t) = v_0 \cdot \mu^t$. For the distance s, that the ball has moved after t steps, it holds:

$$s(t) = \sum_{i=1}^{t} v_0 \cdot \mu^{i-1} = v_0 \cdot \frac{1-\mu^t}{1-\mu}$$

In the sequel, we make the assumption that an agent can move in any direction with a fixed (maximal) velocity v_1. At time $t = 0$, the ball is at position $\vec{a}$ in the Cartesian

coordinate system with the agent at its origin. Let $\vec{b}$ (with $\|\vec{b}\| = v_0$) be the velocity vector of the ball in this coordinate system. Then, after t steps the ball position is:

$$P(t) = \vec{a} + s(t) \cdot \vec{b}$$

Clearly, the agent can reach the ball at any time t with $\|P(t)\| \leq v_1 \cdot t$. Since there is no closed form for t, we apply *Newton's method* in order to find the zeros of $f(t) = \|P(t)\| - v_1 \cdot t$. Therefore, we need the derivative $f'(t) = \|P\|'(t) - v_1$ with

$$\|P\|'(t) = \frac{P'(t) \cdot P(t)}{\|P(t)\|}$$

where in this context $\cdot$ denotes the inner product of vectors. We compute the following sequence t_n, until $|f(t_n)| < \epsilon$ for some small threshold $\epsilon > 0$:

$$t_n = \begin{cases} 0, & \text{if } n = 0 \\ t_{n-1} - \frac{f(t_{n-1})}{f'(t_{n-1})}, & \text{if } n > 0 \text{ and } f'(t_{n-1}) < 0 \\ 999, & \text{otherwise} \end{cases}$$

This procedure eventually yields the first of at most three zeros $t > 0$. There exists at least one zero; it is found at latest after t_n has been set to 999, which avoids oscillation. If there are three zeros, then Newton's method will find the smallest one. This follows from the fact that the acceleration a of the ball (the derivative of v) is negatively proportional to v. A similar (but different) method for computing the interception time has been described in [11]. Figure 1 sketches the situation for ball interception and also gives an example for the function f with three zeros. We see that, after a phase where the ball can be reached by the player, there is a phase where the ball is out of reach, and finally, when the ball has slowed down sufficiently, the ball can be reached again.

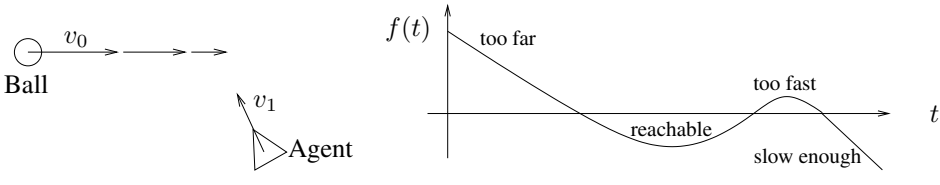


Fig. 1. Reaching the ball

2.3 Applying Reinforcement Learning

An analytical approach to ball interception relies on exact knowledge of system behavior. For a qualitative approach, this might not be necessary, but at least some control knowledge (what action leads to what behavior) must be available. The reinforcement learning

approach described in [8] assumes neither control knowledge nor system knowledge in advance (i.e. at the time of programming). A controller learns a policy for a given goal, and at the time of application the learned policy is used to achieve the goal. However, like in the analytical approach, the assumption is that the underlying model of the world does not change during execution. Reinforcement learning has successfully been used for different problems in simulated robotic soccer [9], and in the subsequent sections we are going to compare a ball interception method trained with reinforcement learning and our methods (see Sect. 4).

3 Ball Interception with Qualitative Velocity

For a brief overview about qualitative reasoning in general, refer [4]. The article comes up with a motivation and some short historical remarks about the field of qualitative reasoning. In that paper, the author gives some real world application examples as well as an introduction into basic qualitative reasoning techniques.

3.1 Approaches with Qualitative Velocity or Trajectories

Representation of motion in a qualitative manner is the subject of the paper [7]. The goal is to represent the trajectory of a moving object and abstract from irrelevant data at the same time. The representation in this paper is based on sequences of qualitative motion vectors, which consist of two components, that is a qualitative direction and a qualitative distance. The qualitative direction component is representing the direction of the movement, while the qualitative distance component depicts the distance the object moved into the given direction. These two components alone are not sufficient to represent the speed of the object. Though it is not in the focus of the paper, it can be found that for a qualitative representation of motion speed one has to use the ratio of time elapsed between two directional changes and the distance. However, for the qualitative representation of motion tracks, speed is not used during the rest of the paper.

An approach to modeling behavior of physical devices qualitatively is described in [2]. Qualitative values are assigned for numerical values and their derivatives. Qualitative variables are used to explain different states of a physical system. The authors explain rules that have to be valid for qualitative variables in order to get a correct description of a system. In this approach, qualitative velocity and also its derivative, the acceleration is considered. However, these values are reduced to one dimension with range $+$, $-$ or 0. By this abstraction, some problems can be described quite adequately, e.g. a pressure regulator and a mass-spring-friction system. However, spatial information is not present in this approach, that is needed for robotic soccer. For further details, the reader is referred to the paper [2]. A more general perspective is taken in [5], where qualitative differential equations are discussed.

The double-cross calculus [12] has been invented to navigate using qualitative spatial information. The double-cross calculus uses a set of three points and a set of 15 base relations to achieve this. The three points (the observer, the point where the observer is looking to and a reference object) are related by one of the relations. It is also possible to represent incomplete information by using unions of relations. Like in the above-mentioned approaches, a static environment is assumed. Hence, this approach is also

not sufficient for highly dynamic environments. Nevertheless, the double-cross calculus has applications in geographical information systems (GIS).

The qualitative approaches just mentioned, first and foremost, are dedicated to the cognitively adequate description of physical reality. But in this context, we want to apply qualitative information to the control of agent behavior. We will see that only qualitative direction and velocity is sufficient for the task of ball interception. One major observation is that the direction to the interception point need not be given too exact (see Sect. 3.2). This is one of the main ingredients for the qualitative approach for ball interception (see Sect. 3.3).

3.2 Qualitative Correction of Direction

The optimal strategy of an agent for ball interception is to first turn its body towards the interception point and then head to this point directly. If the interception point can be computed exactly, then the agent just has to move the distance d to this point, after turning its body once by the angle δ, that is the difference between the current orientation of the agent and the direction to the ball. See also Fig. 2. Thus, the agent could obey the following rule: if the interception point is straight ahead, continue the movement into this direction; otherwise, correct the orientation accordingly.

If there is no inaccuracy in both the sensor data about directions and the actions performed by the agent, then this rule yields an optimal strategy for the agent. In this case, the agent turns only once, namely at the beginning, and then moves straight forward to the interception point. However, in practice (i.e. for real or simulated robots), the direction often cannot be determined exactly. Hence the rule stated above leads to the undesirable behavior that the agent corrects its orientation most of the time, while not coming closer to the interception point.

Therefore, the rule should be relaxed and a correction should only be done, if the angle for correcting the direction exceeds a certain threshold ϵ. This strategy is also illustrated in Fig. 2. The distance $d_1 + d_2$ the agent has to move in this case clearly is longer than d (because of the triangle inequality). Of course, this is a disadvantage, but as the evaluation (in Sect. 4) reveals, the agent does not turn too often and therefore reaches the goal faster.

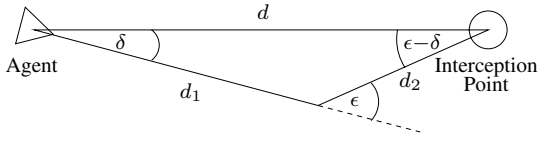


Fig. 2. Correction of direction

The interesting question now is clearly: how much does the distance increase, if the correction of direction is done only after a certain threshold angle ϵ is exceeded? For this, we apply *Mollweide's formula* for triangles with sides a, b and c, and corresponding

angles α, β and γ:

$$\frac{a+b}{c} = \frac{\cos\frac{\alpha-\beta}{2}}{\sin\frac{\gamma}{2}}$$

If we apply this equality to the triangle in Fig. 2, we get the ratio

$$\frac{d_2+d_1}{d} = \frac{\cos\frac{\delta-(\epsilon-\delta)}{2}}{\sin\frac{180°-\epsilon}{2}} = \frac{\cos(\delta-\frac{\epsilon}{2})}{\cos\frac{\epsilon}{2}}$$

which becomes maximal for a given (fixed) threshold angle ϵ, if $\cos(\delta-\frac{\epsilon}{2}) = 1$, i.e. $\delta = \frac{\epsilon}{2}$. Thus in the worst case the ratio is $1/\cos\frac{\epsilon}{2}$. As the following table shows, the overhead wrt. the distance can be neglected for angles up to about $30°$.

threshold angle ϵ	0°	5°	10°	20°	30°	45°	60°	90°
overhead of $1/\cos\frac{\epsilon}{2}$	0.0%	0.1%	0.4%	1.5%	3.5%	8.2%	15.5%	41.4%

Therefore, the threshold can be chosen quite high. This does not decrease the performance of the agent for ball interception significantly. Hence, an approximating, i.e. a qualitative method seems to be very appropriate in this context. We will state a qualitative method for ball interception in Sect. 3.3. Its evaluation is given in Sect. 4.

3.3 A Method with Qualitative Velocity

A naïve method for ball interception would be just to go straight ahead to the ball. This certainly can be seen as a non-numerical, qualitative approach. But since the ball moves, this strategy obviously can be improved. It is better to go to the (earliest) interception point directly. However, since it often cannot be computed exactly, a qualitative approach is preferable. This can be done by making use of qualitative velocity and directions. If the agent looks at the (possibly) moving ball, the agent can distinguish whether the ball moves (quick) to the left or right (from the point of view of the agent), or there is no clear bias to one side. This means we consider the projection of the ball velocity that is orthogonal to the line from the agent to the ball. Note that we abstract from the component of the velocity which is parallel to this line.

As a measure for qualitative velocity we take the following model, which makes the simplifying assumption that the ball moves with a constant velocity. The ball velocity v_0 is normalized wrt. the constant (maximal) velocity v_1 of the agent. Let us now have a look at Fig. 3. The ball moves with velocity v_0 to the point where the agent can intercept it with velocity v_1. The angle β is the angle from the ball between the agent and the (approximated) interception point. The qualitative velocity is now determined by the angle α, that is the angle from the agent between ball and interception point. The agent and the ball must arrive after the same time t at the interception point. Therefore, by applying the law of sine we get

$$\frac{\sin\alpha}{\sin\beta} = \frac{v_0\cdot t}{v_1\cdot t} \quad \text{and hence} \quad \sin\alpha = \frac{v_0\cdot\sin\beta}{v_1}.$$

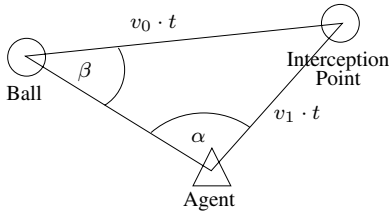


Fig. 3. Determining the velocity qualitatively

Note that α corresponds to the ball velocity v_0 projected to the normal of the line from the agent to the ball, relative to the maximal velocity v_1 of the agent. Thus, it is in fact a qualitative notion of velocity. In order to determine the velocity in an even more qualitative manner, the agent considers a finite number of sectors around the agent. Sectors are also used for qualitative orientation information in [1]. Usually the number of sectors is a power of 2. Let us first investigate the case with $n = 8$ sectors. Then each sector has the size $\varphi = 360°/8 = 45°$. In this context, we map the angle α from above to one of the sectors (as shown in Fig. 4), i.e. to a positive or negative multiple of φ:

$$sign(\alpha) \cdot min(round(abs(\alpha)/\varphi),\, 90°)$$

But since we only approximate the actual movement, we only consider 5 different values for α, namely quick to left ($-90°$), to left ($-45°$), neutral ($0°$), to right ($+45°$), quick to right ($+90°$). Note, that we consider velocity only in one dimension. As before, let δ be the difference angle between the current orientation of the agent and the direction to the ball. Then, the agent just behaves according to the following rule: if $\delta + \alpha \geq \varphi$, then turn by the angle $\delta + \alpha$; otherwise, go straight ahead.

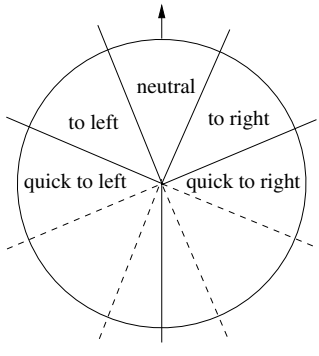


Fig. 4. Qualitative velocity and orientation

4 Evaluation

We conducted several experiments in order to compare the performances of a variety of different approaches to ball interception. Those different methods are the following.

NAI and NAI5 – the naïve methods
If the ball is approximately in front of the player, it runs to the current position of the ball. Otherwise it turns towards the ball. The notion of *approximately in front of* is realized with the help of a *threshold angle* ϵ. If the absolute value of the angle δ between the agent's orientation and the current ball position is less than ϵ, the agent just runs forward (see Fig. 5 left). If, however, $|\delta| > \epsilon$, the agent turns by δ, in order to face the current ball position (Fig. 5 right). Please note, that this method completely ignores the speed and direction of the ball movement.
Following the considerations in Sect. 3.2, we chose 22.5° as value for ϵ in NAI. The method NAI5 differs from NAI in the value of the threshold angle ϵ only. In this method the value of ϵ is set to 5°. Choosing between turning and running is necessary because the Soccer Server treats those actions as mutually exclusive, i.e. in one simulation step a player can either run (dash) or turn.

Fig. 5. Naïve ball interception: $\delta_1 \leq \epsilon \rightarrow$ run; $\delta_2 \geq \epsilon \rightarrow$turn

NWT – Newton's method
Newton's method calculates the point, where the agent can intercept the ball directly. This method has already been described in Sect. 2.2.

LRN – the learned method
This is an approach that uses reinforcement learning to train an agent to intercept the ball. The actual behavior stems from the *Karlsruhe Brainstormers* RoboCup team [9] and has been introduced in Sect. 2.3.

Q8 and Q16 – the qualitative methods
These are two instances of the qualitative method introduced in Sect. 3.3. Q8 uses eight different qualitative directions, while Q16 makes use of sixteen sectors.

4.1 The Setting

We used an automated coach agent – called *evalcoach* – to run the evaluation sessions. A coach is a special type of agent that can connect to the Soccer Server. In contrast to the usual agents (the players) a coach is able to move the ball and players on the field and

to act as a referee, e.g. by changing the play mode of the simulator, e.g. from *play_on* to *free_kick*. See [3] for further details. As this client is intended for evaluation or learning scenarios, it receives data about all the objects on the field in external coordinates, in contrast to the players, that only receive incomplete information in an egocentric coordinate system. In addition to that no noise is added to the data received by the coach client. The basic setup of the evaluation is shown in Fig. 6. A coach and one player are connected to the Soccer Server.

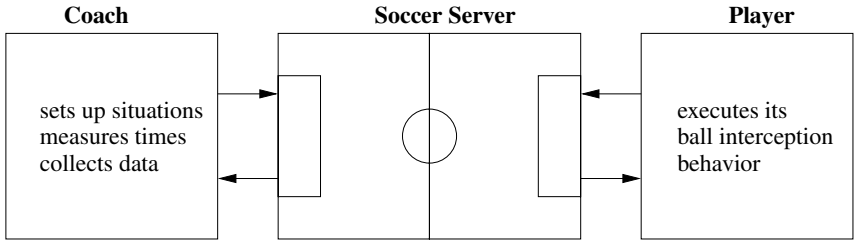


Fig. 6. The evaluation setup

The player is only executing its ball interception behavior. For each of the interception methods described above such a specialized agent exists. In order to reduce the effects of noise and randomization as far as possible, the players made use of a special type of information sent by the Soccer Server, the so called *fullstate*. The fullstate mode allows an agent to get precise information about the position of the ball even if the ball is currently not seen. Similarly the player's position can be determined. In addition to that, noise in the movement of both ball and player have been set to zero. Although it certainly is interesting to compare the different methods in the presence of sensor noise (which could support the claim that qualitative approaches are more robust), the problem is that then the variances of the measurements become too high in the evaluation.

Each evaluation session consists of 250 different *scenarios*. Each scenario is specified by the initial position and velocity of the ball and the player, which are selected according to the following rules:

1. set the agent in the center of the field, with random orientation and speed;
2. place the ball at an arbitrary position within a $40 \times 20\,\mathrm{m}^2$ rectangle in the middle of the field (see Fig. 7); assign a motion with random direction and speed to it.

An *episode* within such a scenario corresponds to the player trying to intercept the ball. It ends, if either the agent has successfully intercepted the ball (i.e. the ball can be kicked by the agent), or the ball goes out of bounds, or a timeout has been reached. The latter two outcomes of an episode are counted as failures. In order to compensate network problems and process load, we ran 100 episodes for each scenario. The control of a session was done by the evalcoach with this algorithm:

while there is another scenario **do**
  read next scenario
  for i := 1 to 100 **do**
    setup scenario
    run episode i
    save duration of episode i
    note *Success or Failure*
  end for
  Record average and variance of duration
  Record percentage of successes
end while

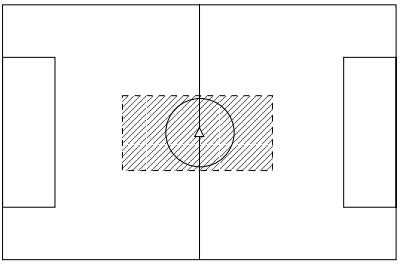


Fig. 7. Initial setup of the evaluation. The agent is in the center of the field. The ball is randomly placed within the hatched rectangle

4.2 Analysis of the Data

For each of the six interception methods described above an evaluation session was done. Thus, we obtained tables containing the average interception time for each scenario, the variances of the times and the percentage of successful episodes per scenario for each method. Impossible scenarios, i.e. scenarios in which *no* player succeeded in intercepting the ball even once, were removed from the table. After this cleaning step, 234 scenarios were left, in which at least one interception method was successful. They built the basis for comparing the different approaches.

Let us present some general figures first. The success rate for each scenario lay by either 100% or 0%. As we eliminated randomness from the evaluation sessions, this means, that no significant disturbances due to network or process load appeared. Nevertheless, influences by the hardware can be observed, as the variances of several scenarios are larger than zero throughout all six sessions.

All interception methods succeeded in almost all scenarios, which can be seen on the left in Fig. 8. On the right side of this figure the percentage of sessions that were "won" by a single method is shown. By "won" we mean, that one method was faster than every other method in the scenario. The slice labeled *rest* summarizes all scenarios that had no unique winner, i.e. where two or more methods were equally good.

Approach	No. of Successes
NAI	232
NAI5	231
LRN	233
Q8	232
Q16	232
NWT	234

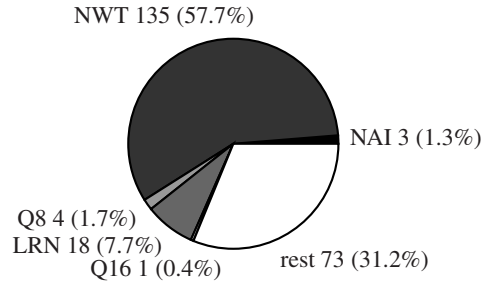


Fig. 8. Left: The number of successful interceptions by each approach. Right: The number of times each method was better than *all* others. In 73 scenarios at least two methods were equally good

Both the number of successes and the number of scenarios won indicate, that Newton's method (NWT) is slightly superior to the other approaches to ball interception. This is supported by calculating the means of the interception times, which are shown in Fig. 9. The lowest average duration belongs to NWT. But all three figures show, that the learned method (LRN) is only slightly worse than NWT.

The naïve method NAI5 clearly is the worst approach to ball interception. It has the lowest number of total successes (231), does not win a single scenario and on average takes the longest times to intercept a ball (31.5 simulation steps). Surprisingly the second naïve method (NAI) does quite well. On average it performs better than both variations of the qualitative method, although Q8 wins more scenarios than NAI.

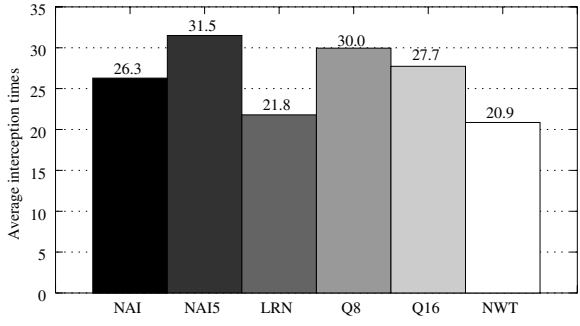


Fig. 9. Average interception times of the different approaches

In addition to the measurements described above, we took the average ratios $\overline{R}$ between interception times. The average ratio $\overline{R_{k,l}}$ of the interception times of methods k and l is given by

$$\overline{R_{k,l}} = \frac{1}{n} \sum_{i=1}^{n} \frac{d_i^k}{d_i^l}$$

where n denotes the total number of scenarios and d_i^j is the interception time of method j in scenario i. Figure 10 shows the average ratios between all pairs of methods. This table once again supports the results from the measurements above, namely a slight superiority of NWT, but which is closely followed by LRN, although LRN seems to do better according to the ratio $\overline{R_{\text{NWT,LRN}}} = 1.005$. A direct comparison between the interception times of LRN and NWT shows, that NWT is faster than LRN 144 times, LRN beats NWT 21 times, and 69 times both methods produce equal interception times. This is also shown on the left in Fig. 11. But if LRN is faster than NWT in a scenario, it is usually *much* faster (up to 15 simulation steps), which has a great impact on calculating the mean of the quotients.

	NAI	NAI5	LRN	Q8	Q16	NWT
NAI	—	0.832	1.436	0.981	1.014	1.413
NAI5	1.261	—	1.835	1.229	1.269	1.807
LRN	0.820	0.696	—	0.759	0.802	1.035
Q8	1.190	0.972	1.624	—	1.105	1.626
Q16	1.077	0.881	1.486	0.961	—	1.472
NWT	0.790	0.670	1.005	0.743	0.777	—

Fig. 10. Average ratios between the methods

Let us now take a closer look at the methods, that we evaluated in two variations, namely NAI and NAI5 as well as Q8 and Q16. In both methods one variation proved to be clearly superior to the other.

NAI versus NAI5. The method NAI beats NAI5 by far. Obviously, this comes from the different values of the threshold angles. As this angle is much smaller in NAI5, the player has to turn often, because the ball moves out of the sector defined by the threshold angle. For a player using NAI for interception, this sector is much wider and thus the need for turning arises less frequently. If the threshold angle is not too wide, the additional way the agent has to run, does not matter very much, as we have shown in Sect. 3.2. Thus, a player using NAI for interception can get to the ball much faster than a player using NAI5.

Q8 versus Q16. The results of the evaluation show, that from the two variations Q8 and Q16 of our qualitative interception method Q16 is the faster one. Although both approaches are successful in the same scenarios, the average interception time is less for Q16. The direct comparison, shown on the right in Fig. 11, shows that Q16 is faster than Q8 in 166 different scenarios, which roughly corresponds to 71% of all scenarios. This result shows that the number of sectors, i.e. qualitative directions, has a significant influence on the performance of the qualitative ball interception. In Q16 the sectors are smaller than in Q8 and thus give the player a more finely grained control over its turns than Q8.

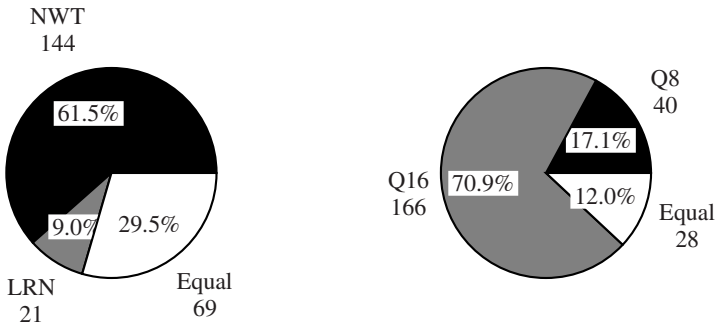


Fig. 11. Direct comparisons between different interception methods. Left: LRN vs. NWT. Right: Q8 vs. Q16

One phenomenon, that can repeatedly be observed, is an oscillation at the beginning of an episode. The player just keeps turning back and forth for some time before it starts to run. The agent decides to turn based on the qualitative position and velocity of the ball, but then it turns just too far. So, in the next step the player decides, that it has to turn back and now turns too far into the other direction. Clearly this effect can be reduced, if the player divides its surroundings in more sectors and thus has more directions to turn to at its disposal.

5 Conclusions

In this paper we presented a variety of methods to intercept the ball in the virtual environment of the RoboCup simulation league. We introduced a qualitative approach to ball interception that makes use not only of qualitative directions, but of qualitative descriptions of velocity as well.

We conducted a number of experiments in order to compare two variations of this approach to several other methods, including a numerical method and a learned behavior. These experiments have shown the numerical method and the learned behavior to be the best. The differences between those two approaches are very small but show a slight advantage of the numerical method.

5.1 How Useful Is Qualitative Velocity?

The qualitative interception methods did not do very well in the evaluation and turned out to be similar in performance to a naïve approach to ball interception which discards all information about the movement of the ball. This is, of course, not that surprising, since a qualitative method works with a rough approximation of the physical reality. Hence, one clear advantage the qualitative interception method has over the numerical and the learned methods is its robustness and portability. For the learned method to work it is necessary that the environment does not change after the learning period. Otherwise

the whole behavior has to be trained again. The numerical method even depends on the complete knowledge of the physical model that describes movements in the environment.

In the simulated environment of the Soccer Server with its simple model of mechanics and the quantitative information about objects received by the clients the abstraction from quantitative data onto a qualitative level seems somehow artificial, as the agent has to abstract from quantitative information provided by the simulator, reason on the qualitative data and finally map qualities back to concrete numbers when it sends a command. But consider a domain which is too complex to be described quantitatively or in the level of detail needed by the quantitative methods. In other (real world) domains it may be very expensive or slow to generate quantitative data from sensor inputs but relatively easy to achieve a qualitative representation of the world. As the qualitative interception method makes only very few basic assumptions about the mechanics involved – e.g. the ball motion does not change its direction without external influences – it is not hampered by complexity of the underlying physics, e.g. several kinds of friction and noise in the sensor data.

Furthermore, the representation of the ball position and movement as well as the actions taken by the player are more cognitively adequate. At least, they are more symbolic and propositional, such that the representations are easy to understand and calculate. Real-time requirements are easily met because of the simplicity of the calculations. We only roughly have to measure the velocity (projection) of the ball.

5.2 Future Work

Future work includes further analysis of the data collected in the evaluation sessions. Since no method was always fastest, but rather every method was better than the others in one scenario or another, we hope to gain more information about the strengths and weaknesses of the individual interception methods by further examining and classifying the data and the different scenarios.

Another piece of work, that will be tackled soon, is the optimization of the parameters of our qualitative interception method. As we have seen, the number of different directions the agent knows, have a significant influence on the performance of the method. In parallel to finding the optimal number of sectors for the proposed method, we will examine the influence of adding methods for reducing the oscillation effect on the overall performance of the method.

Last but not least, we plan to apply the proposed method to other domains in order to test its scalability to other applications (possibly with sensor noise) and its feasibility outside the world of the Soccer Server. This might help us to understand better where a qualitative approach for spatial reasoning with velocity is appropriate – not (only) for the qualitative description of physical behavior, but also for controlling agents.

Acknowledgments. We would like to thank Marion Levelink, Artur Merke, Reinhard Moratz, Martin Riedmiller and Manfred Stolzenburg for helpful comments, hints to relevant work, and helping us with the evaluation.

References

1. Eliseo Clementini, Paolino Di Felice, and Daniel Hernández. Qualitative representation of positional information. *Artificial Intelligence*, 95(2):317–356, 1997.
2. Johan de Kleer and Daniel G. Bobrow. Qualitative reasoning with higher-order derivatives. In *Proceedings of the American National Conference on Artificial Intelligence (AAAI-84)*, pages 86–91, 1984. Reprint in Daniel S. Weld and Johan D. Kleer (eds.), *Readings in Qualitative Reasoning about Physical Systems*, Morgan Kaufmann, San Francisco, 1990.
3. Ehsan Foroughi, Frederik Heintz, Spiros Kapetanakis, Kostas Kostiadis, Johann Kummeneje, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, and USTC9811 Group. *RoboCup Soccer Server User Manual (for Soccer Server Version 7.06 and later)*, 2001.
4. Yumi Iwasaki. Real-world applications of qualitative reasoning. *IEEE Expert*, 12(3):16–21, 1997.
5. Herbert Kay, Bernhard Rinner, and Benjamin Kuipers. Semi-quantitative system identification. *Artificial Intelligence*, 119(1-2):103–140, 2000.
6. Jan Murray, Oliver Obst, and Frieder Stolzenburg. Towards a logical approach for soccer agents engineering. In Peter Stone, Tucker Balch, and Gerhard Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, LNAI 2019, pages 199–208. Springer, Berlin, Heidelberg, New York, 2001.
7. Alexandra Musto, Klaus Stein, Andreas Eisenkolb, Thomas Röfer, Wilfried Brauer, and Kerstin Schill. From motion observation to qualitative motion representation. In Christian Freksa, Wilfried Brauer, Christopher Habel, and Karl F. Wender, editors, *Spatial Cognition II*, LNCS 1849, pages 115–126. Springer, Berlin, Heidelberg, New York, 2000.
8. Martin Riedmiller. Concepts and facilities of a neural reinforcement learning control architecture for technical process control. *Journal of Neural Computing and Application*, 8:323–338, 2000.
9. Martin Riedmiller, Artur Merke, D. Meier, A. Hoffmann, A. Sinner, O. Thate, C. Kill, and R. Ehrmann. Karlsruhe Brainstormers – a reinforcement learning way to robotic soccer. In Peter Stone, Tucker Balch, and Gerhard Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, LNAI 2019, pages 367–372. Springer, Berlin, Heidelberg, New York, 2001.
10. Peter Stone et al. Robocup-2000: The fourth robotic soccer world championships. *AI magazine*, 22(1):11–38, 2001.
11. Peter Stone and David McAllester. An architecture for action selection in robotic soccer. In *Fifth International Conference on Autonomous Agents*, 2001.
12. Kai Zimmermann and Christian Freksa. Qualitative spatial reasoning using orientation, distance, and path knowledge. *Applied Intelligence*, 6:49–58, 1996.

Spatial Inference – Learning vs. Constraint Solving

Carsten Gips, Petra Hofstedt, and Fritz Wysotzki

Berlin University of Technology
{cagi,ph,wysotzki}@cs.tu-berlin.de

Abstract. We present a comparison of two new approaches for solving constraints occurring in spatial inference. In contrast to qualitative spatial reasoning we use a metric description, where relations between pairs of objects are represented by parameterized homogenous transformation matrices with numerical (nonlinear) constraints. We employ interval arithmetics based constraint solving and methods of machine learning in combination with a new algorithm for generating depictions for spatial inference

1 Introduction

Understanding and interpretation of textual descriptions of real world scenes are important for many fields, e.g. navigation and route descriptions in robotics [13,15], in CAD or in graphical user interfaces (e.g. "The xterm is right of the emacs.").

In contrast to qualitative approaches to spatial reasoning [6,8], in [3] we presented a new metric approach to spatial inference based on mental models ([3, 12]). Starting from sentences like "The lamp is left of the fridge." we try to create a mental model which represents the described spatial situation. This approach uses a directed graph, where the nodes represent the objects and the edges represent the given relation, e.g. `left(fridge, lamp)`, between two objects. From this model it is possible to infer relations which were not initially given in the text or to generate depictions compatible with the description.

The semantics of the relations is given by homogenous transformation matrices with constraints on the variables. As shown in [16], inference of a relation between two objects is done by searching a path between the objects and multiplying the matrices on this path. Thereby constraints containing inequalities and trigonometric functions must be propagated and verified. Only in some rare cases we can solve these constraints analytically. Furthermore in [16] a simple algorithm for generating depictions is proposed. It is restricted to default positions of objects and to rotations of multiples of $\pi/2$. Moreover, this approach requires to keep lists with possible positions for every object.

Our aim is now to find a method to solve this kind of constraints and to generate depictions without the restrictions mentioned above. In this paper we sketch two approaches to spatial reasoning: First, we use cooperative constraint

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 299–313, 2002.

solving methods, in particular interval arithmetics, where inference is supported by further solving methods. Similarly, in [5] the constraint solver Parcon has been integrated into a 2D real-time animation environment. While [5] aims at one particular solution for the placement of objects, our interest is on the whole solution space. Moreover, we will see that our approach is more flexible because our system allows the integration of new solvers, like Parcon itself, in a simple way. This is advantageous if new requirements to the problem formulation appear. Our second approach on spatial reasoning applies machine learning in combination with a new algorithm for depiction generation.

This work is structured as follows: We start with an introduction into the description of spatial relations by means of examples in Sect. 2. In Sect. 3 we demonstrate an approach of directly solving spatial constraints with cooperating constraint solving methods. An alternative approach is to use machine learning as described in Sect. 4. In Sect. 5 we compare the different approaches wrt. their advantages and disadvantages and show perspectives for future work.

2 Expressing Spatial Relations

Our aim is to describe scenes with the help of spatial relations and, given such spatial descriptions, to generate appropriate scenes or to find out that no actual scene according to the current description exists, resp. For simplification, we consider only 2D scenes and represent objects by appropriate geometric figures.

Example 1. Given the spatial relations `right(cupboard, lamp)` and `left(fridge, lamp)` a possibly intended scene is given in Fig. 1.

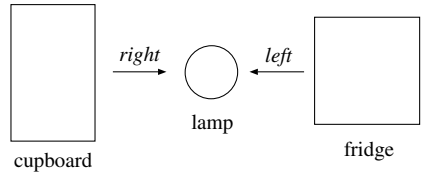


Fig. 1. Example scene 1: `right(cupboard, lamp)` and `left(fridge, lamp)`

However, this is not the only possible scene. The question, whether a scene is an actual representation of the given set of relations, depends of course on the intended meaning of the relations. For example, the relation `right(cupboard, lamp)` does not necessarily describe only a situation, where the lamp is straight right from the cupboard. The lamp could be situated for example downright or upright from the cupboard as well.

For our purposes we investigated scene descriptions based on the relations `left/2` and `right/2`, which describe the placement of an object left resp. right from another one, the relations `front/2` and `behind/2` which place objects in

front of or behind other objects and the relation `at_wall/2` for describing the placement of an object parallel to a wall with a fixed maximum distance. Further relations provide that an object is situated in a given room (`in_room/1`) and they ensure, that objects do not overlap (`not_overlap/n`).

As mentioned above, in contrast to qualitative techniques [6,8] for spatial reasoning we use a metric approach [3,4] known from the area of robotics ([2]). At this, we associate with every object a coordinate system, its form and size. Relations between pairs of objects are represented by their transformation matrices. Thus, the current coordinates of an object depend on its relation, i.e. orientation and distance, to its relatum, which may be different in different constraints. That means, changing the relatum of an object we need to transform its coordinates using the corresponding matrix.

Let us consider the relation `right/2` in detail. The relation `right(cupboard, lamp)` places the lamp, which is the referent, right wrt. the cupboard, its relatum. The cupboard is the origin of the relation. The lamp as referent can be placed within the bisectors of the right angles of the cupboard. Figure 2 illustrates this situation: The lamp (represented by the circle) is placed `right` of the cupboard (represented by the rectangle).

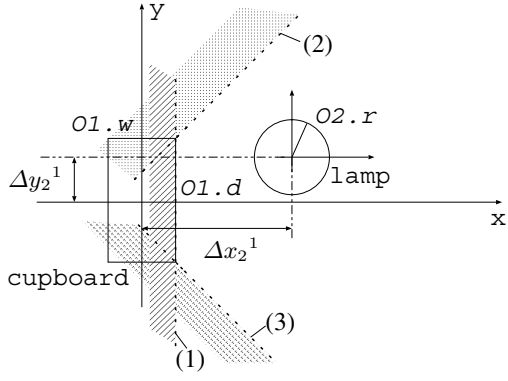


Fig. 2. The relation `right(cupboard, lamp)` in detail

Mathematically we can describe the relation `right(`$O1$`, `$O2$`)` ($O1$ and $O2$ stand for the cupboard and the lamp, resp.) by the following inequalities:

$$\Delta x_2{}^1 \geq O1.w + O2.r \qquad (1)$$
$$\Delta x_2{}^1 \geq \Delta y_2{}^1 + O1.w - O1.d + \sqrt{2}O2.r \qquad (2)$$
$$\Delta x_2{}^1 \geq -\Delta y_2{}^1 + O1.w - O1.d + \sqrt{2}O2.r \qquad (3)$$

At this, $O1.w$ and $O1.d$ represent the width and the depth of the rectangle, i.e. the cupboard, and $O2.r$ stands for the radius of the lamp. The distances of the object $O2$ in the x- and y-directions from the relatum $O1$ are denoted by

$\Delta x_2{}^1$ and $\Delta y_2{}^1$, resp. At this, the lower index is associated to the referent and the upper index to the relatum.

Note, that for the relation `right/2`, like for every spatial relation, in general the formulae differ depending on the form of the relata and referents.

In the remainder of this paper we will use the following example to demonstrate our work.

Example 2. We extend Example 1 giving further constraints which describe the sizes of our objects and a room. We require the objects to be inside the room and not to overlap each other. A corresponding scene is given in Fig. 3.

`room.w` $\in$ `[4.0, 4.5]`, `room.d` $\in$ `[4.0, 4.5]`, `lamp.r = 0.3`,
`fridge.w` $\in$ `[0.4, 0.5]`, `fridge.d` $\in$ `[0.4, 0.5]`
`cupboard.w = 0.4`, `cupboard.d` $\in$ `[1.0, 1.2]`,
`left(fridge, lamp)`, `right(cupboard, lamp)`,
`in_room(fridge)`, `in_room(lamp)`, `in_room(cupboard)`,
`not_overlap(fridge, lamp, cupboard)`

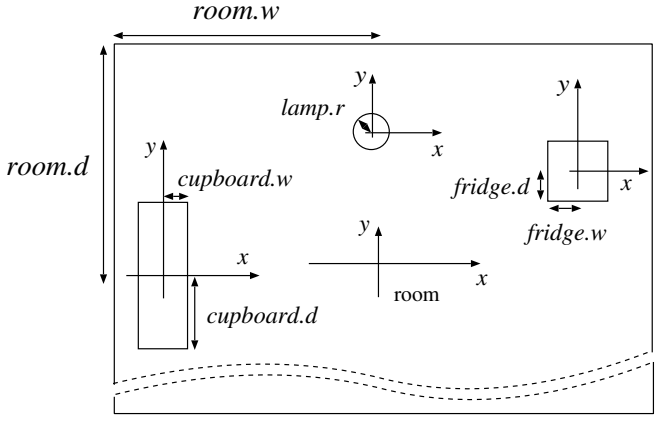


Fig. 3. Example scene

3 Constraint Solving

An approach to reason about such kind of spatial knowledge is to directly use constraint solvers appropriate for this kind of problem.

Representing spatial situations requires interval constraints, because the sizes of the objects and the room are often not given in detail, and we need further

arithmetic constraints. Thus, for this application it is appropriate to use an interval arithmetic solver, like the Brandeis Constraint Solver [1,9], which handles basic operations, like addition and multiplication on rational intervals, as well as trigonometric and logarithmic functions.[1] Note that this solver is incomplete, which means, that it does not detect every unsatisfiable constraint. Thus, it could be useful to employ a further solver for a part of the constraints. Since we are interested in getting particular scenes, i.e. generating depictions, we, moreover, would like to use a constraint solver for guessing the placement of objects. Thus, the cooperation of different constraint solvers is desirable here.

In [10] a general scheme for the cooperation of different constraint solvers is proposed, in [11] an according implementation is shortly sketched. The problem of handling spatial knowledge is a typical application for this system.

The cooperating system allows to integrate arbitrary black box solvers providing a typical solver interface. A meta mechanism coordinates the work of the individual solvers and the information exchange between them. The system can be configured by the user wrt. strategies and for evaluation of experiments. It is possible to define a wide range of different cooperation strategies according to the current requirements. Using this cooperating system, it is possible to deal with hybrid constraints over different constraint domains, and thus to describe and to solve problems which none of the single solvers can handle alone.

An input file for the implementation (see [11] for details) describing the problem of Example 2 is given in Fig. 4. In the first line in the `[solver]` part we specify the constraint solver to be used: the Brandeis Interval Arithmetic Solver `ISolver`.[2]

In the file, `O1` stands for the cupboard, and `O2` and `O3` represent the lamp and the fridge, resp. Instead of $\Delta x_2{}^1$ we write `dx21`. In the `[constraints]` part we give constraints describing the sizes of the room and our objects (lines (1)-(4)) and the constraints which (quantitatively) describe the relations `right(cupboard, lamp)` (lines (7)-(9)) and `left(fridge, lamp)` (lines (12)-(14)). Further constraints (lines (5)-(6) and (10)-(11)) express a transformation of object coordinates because of changes of the relata and/or referents. Constraints for placing the objects in the room and for ensuring that the objects do not overlap are left out here.

Our system computes for this input file the following solution space:

$$\begin{array}{rcl} 4.0 \leq \texttt{room.w} \leq 4.5 & \qquad & 4.0 \leq \texttt{room.d} \leq 4.5 \\ -4.1 \leq \texttt{dx10} \leq 2.7 & & -3.5 \leq \texttt{dy10} \leq 3.5 \\ -3.4 \leq \texttt{dx20} \leq 3.4 & & -4.2 \leq \texttt{dy20} \leq 4.2 \\ -2.7 \leq \texttt{dx30} \leq 4.1 & & -4.1 \leq \texttt{dy30} \leq 4.1 \end{array}$$

[1] For reasons of simplicity, we do not handle trigonometric nor logarithmic functions in this paper. Nevertheless they can be used, for example, to describe rotations of objects in our approach.

[2] Configurations of the used solvers and the solving strategy are allowed to be given in the input file by the user. Here they are left out.

```
[solver]
  ISolver  = solver.brandeis.Brandeis
[constraints]
  room.w, room.d in [4, 4.5];        # width and depth of the room    (1)
  O1.w = 0.4; O1.d in [1.0, 1.2];    # size of the cupboard           (2)
  O3.w, O3.d in [0.4, 0.5];          # size of the fridge             (3)
  O2.r = 0.3;                        # radius of the lamp             (4)
           # the coordinates of the lamp wrt. the cupboard
  dx21 = dx20 - dx10;                                                 (5)
  dy21 = dy20 - dy10;                                                 (6)
           # the relation right(cupboard, lamp):
  dx21 >= O1.w + O2.r;                                                (7)
  dx21 >= dy21 + O1.w - O1.d + (2 ^ 0.5) * O2.r;                      (8)
  dx21 >= - dy21 + O1.w - O1.d + (2 ^ 0.5) * O2.r;                    (9)
           # the coordinates of the lamp wrt. the fridge
  dx23 = dx20 - dx30;                                                (10)
  dy23 = dy20 - dy30;                                                (11)
           # the relation left(fridge,lamp):
  dx23 <= - O3.w - O2.r;                                             (12)
  dx23 <= dy23 - O3.w + O3.d - (2 ^ 0.5) * O2.r;                     (13)
  dx23 <= - dy23 - O3.w + O3.d - (2 ^ 0.5) * O2.r;                   (14)
```

Fig. 4. Constraint file describing Example 2

At this, `dx10`, `dx20`, and `dx30` represent the distance of the objects `O1`, `O2`, and `O3` resp. from the coordinate origin of the room as relatum. The computed solution space restricts the space for placing the objects. Figure 5 shows the area (shaded) for a possible placement of the cupboard (i.e. its center (`dx10, dy10`)) in the room. The depth and width of the room (`room.d` and `room.w`) are between 4 and 4.5 length units. The cupboard can be placed on the left wall (`dx10 = -4.1`) of the room (of width 4.5). Because of the relations `right(cupboard, lamp)` and `left(fridge, lamp)`, it cannot be situated on the right wall, there must be some distance for placing the other objects, thus `dx10` $\leq$ `2.7` holds. For `dy10` holds $-3.5 \leq$ `dy10` ≤ 3.5, this is due to the depth `O1.d` of the cupboard.

However, we are not only interested in restricting the solution space of the variables, but we also want to derive particular depictions. Thus, we introduce constraints which propose possible positions of the objects and a solver to handle such constraints.

Besides the interval arithmetic solver `ISolver` we use now a finite domain constraint solver `FDSolver` for the computation of depictions. A finite domain solver is able to handle constraints over finite domains, in particular constraints over finite sets. In our case, this is used to handle (finite) sets which describe possible object locations. The corresponding constraints of the `FDSolver` are given in Fig. 6. While the constraints of the lines (1)-(14) as given in Fig. 4 are constraints of the interval solver `ISolver`, the set constraints of lines (15) and

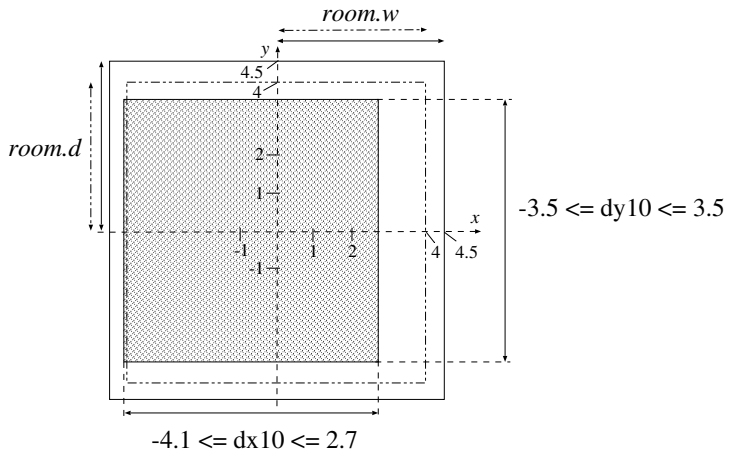


Fig. 5. Possible positions for a placement of the cupboard `O1` in the room

```
[solver]
  FDSolver = solver.csplib.CSPlib
  ISolver  = solver.brandeis.Brandeis
[constraints]
  ...
  dx10, dx20, dx30 inn {-3.0, 0.0, 3.0};                    (15)
  dy10, dy20, dy30 inn {-1.5, 1.5};                         (16)
```

Fig. 6. Extension of the constraint file for generating depictions

(16) in Fig. 6 are constraints of the finite domain solver `FDSolver`. Both solvers understand equality constraints between variables and ground values like the constraint `O1.w = 0.4` of line (2) and the constraint of line (4) which are, thus, assigned to both solvers.

Computing solutions, the cooperating system delays the generation of depictions by the finite domain constraint solver to avoid the splitting of the solution space as long as possible. This principle, known from other systems, e.g. [7], avoids unnecessary computations caused by nondeterministic search.

Now, the computation yields the 4 solutions represented in Fig. 7. Looking at these depictions as solutions, we see that the objects are always placed in the sequence cupboard – lamp – fridge. This is due to the constraints `right(cupboard, lamp)` and `left(fridge, lamp)`. For the fridge and the lamp the `dy`-values are always the same while the `dy`-value of the cupboard may differ from that. The reason is that the lamp can be placed within the bisectors of the left angles of the fridge only and a depiction with different `dy`-values for the lamp and the fridge is, thus, no solution. The cupboard has a larger depth and thus

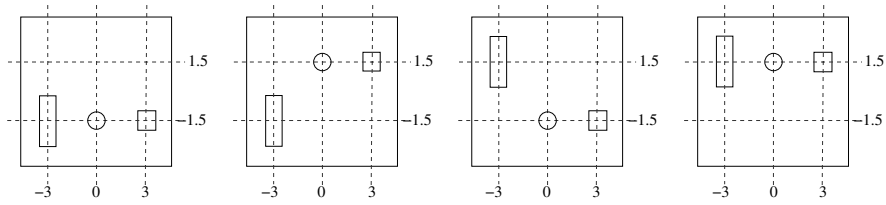


Fig. 7. Generated depictions

the bisectors of its right angles include the placement of the lamp at another `dy`-value as well.

Even if we can describe our problem in this way, and we are able to compute depictions, specifying the problem using the pure mathematical description is not very comfortable and causes the user to write a large number of constraints. To get nearer to a human description, it is possible to integrate a new solver which simply performs macro rewriting for particular constraints, for example it replaces a constraint `right(cupboard, lamp)` by the corresponding mathematical constraints (lines (5)-(9) in Fig. 4). In this way the number of constraints which the user must provide can be decreased and the readability is improved.

Using the described system of cooperating solvers allows to specify spatial problems in an explicitly quantitative way and as well by introducing translation functions in a more natural way. It is possible, on the one hand, to restrict the solution spaces and, on the other hand, to compute depictions, i.e. example scenes which satisfy a given description, by introducing a finite domain constraint solver and appropriate constraints. Times for computing solutions are acceptably of a fraction of a second. The system allows to introduce new black box solvers according to the current problem. If the kind of problem changes or there are new requirements to the problem formulation, the integration of new constraints and new appropriate solvers is possible in a simple and comfortable way.

However, the approach has two disadvantages: At first, the available solvers are often incomplete. That means, that a constraint solver is not able to detect every unsatisfiable constraint. Incompleteness is due to the underlying constraint domain and the associated solving algorithms. Even if, due to information exchange between the solvers, the cooperating system is able to detect inconsistencies of a given constraint conjunction which the individual solvers are not able to detect, in general, the incompleteness of the individual solvers is taken over by the overall system. The integration of further constraint solvers, for example, a solver for linear arithmetics based on the simplex method, could allow to stronger restrict the solution space, however it does not actually solve the incompleteness problem in general. The other problem is, that with increasing complexity of the spatial scene and with an increasing number of objects and, thus, of constraints and variables the computations become more time consuming.

Thus, in the following Sect. 4 we consider another but similar approach for describing and solving spatial scenes which is as well promising. In Sect. 5 we discuss their relations and a possible interaction.

4 Machine Learning

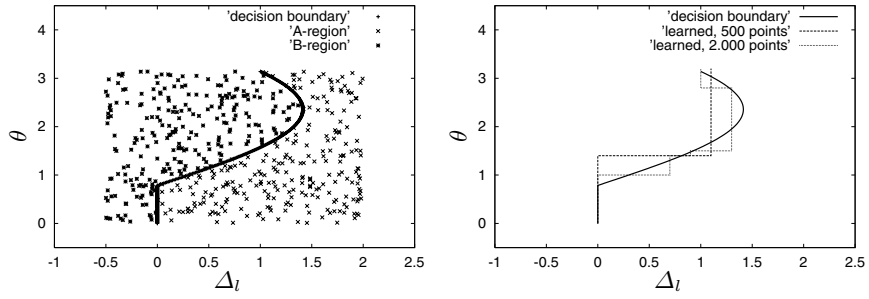
Instead of solving the constraints directly we try to learn the decision function $A(x_1, \ldots, x_n)$ which decides whether a vector $x = (x_1, \ldots, x_n)$ of the configuration space belongs to a region where the predicate A is true, i.e. the corresponding constraints are satisfied. In our problem domain (objects located in a room), the constraints consist of equations and inequalities containing trigonometric functions which lead to computational difficulties well known from robotics [2].

Before employing machine learning algorithms, we have to construct a training set by exploiting the given constraint description or by using results of psychological studies. These datasets consist of preclassified feature vectors where each variable of the constraints represents an attribute, i.e. a dimension in the feature space. In the following, we will use "class A" for the regions where a constraint A is satisfied (and "class B" otherwise). By means of the training sets algorithms of classification learning (e.g. decision tree learning like CAL5 or neuronal nets like Dipol, see [14] for both) construct classifiers now. These decide the class membership of an arbitrarily chosen point x (not necessarily contained in the training set) by inductive generalization. This decision is very fast in comparison to using the system of equations and inequalities for a direct computation of the class membership and therefore it is especially suited for online tasks. In addition, the decision rules make the regions of the configuration space, where the constraints are satisfied, (approximately) explicit.

Generating training sets in order to get an acceptable approximation of the decision boundary is also known as "learning by exploration" or "active learning" in literature and is subject of current research ([17]).

4.1 Learning Spatial Relations with CAL5

Decision tree learners approximate the class boundaries piecewise linearly by axis-parallel hyperplanes. Usually, there is a generalization error due to the unavoidable approximation of the boundaries between the A-regions and the B-regions. This error can be measured using a test set of classified example vectors different from the training set. By increasing the number of training data (and simultaneously shrinking a certain parameter of CAL5) the generalization error can be reduced (i.e. the accuracy of the class boundary approximation can be made arbitrarily high), and in the limit of an infinite training data set the error becomes zero. This is shown in the following example, which is taken from [4]. There we learned a constraint $\sin\theta = \Delta_l/2 \pm \sqrt{1/2 - \Delta_l^2/4}$, which expresses the relation "A bar is right or behind of an object O". The point S is the origin of the coordinate system of object O. Thus neither O nor S occur in the inequality. Δ_l is the difference of the displacement of a particular point of the bar in x-direction and y-direction of S divided by the length of the bar.



(a) A-region and *computed* boundary (b) Boundaries for 500 and 2.000 points

Fig. 8. Learned vs. computed boundaries

Figure 8(a) shows the A-region with computed boundary vs. the B-region. In Fig. 8(b) the solid real (computed) boundary of the A-region is compared with the dashed learned boundaries for a training set of 500 and 2.000 points, resp. It can be seen that the generalization error of the obtained decision tree shrinks with an increasing number of points for learning. However, in practice we reach the manageable limit at 200.000 training examples. The constraints of our relations (like `right/2` for `circles` and `rectangles`, see Sect. 2) affect up to seven parameters, thus, we obtain a configuration space of up to seven dimensions. Thereby we get approximatively ten data points per dimension[3] in average. Because of this sparsely populated configuration space, both the training and generalization errors are rather high. This is shown in Table 1, where we used 200.000 uniformly distributed data points for learning and 5.000 points for testing.

Table 1. Results of the learning process for some spatial relations

Relation[4]	Number of	Points		Test error		
	class A leaves	in A	in B	for A only	for B only	overall
`atwall(r, r)`	143	15.909	184.091	10 %	1 %	2 %
`front(c, r)`	1.984	84.285	115.715	22 %	14 %	17 %
`right(c, r)`	1.702	84.557	115.443	20 %	13 %	16 %
`right(r, c)`	2.079	87.550	112.450	26 %	14 %	19 %

[3] Supposed we obtain the same number of data on each dimension (like a grid), the 7th root of 200.000 yields approximately six data points on each dimension. Note, this is not a correct calculation but a simple estimation.

Furthermore we have to take care for a sufficient large subspace of configuration space. The result of a too small scope of training data vs. the area of the example room is shown in Fig. 9. The resulting classifier does not cover the intersection of the room area and the `right/2` sector.

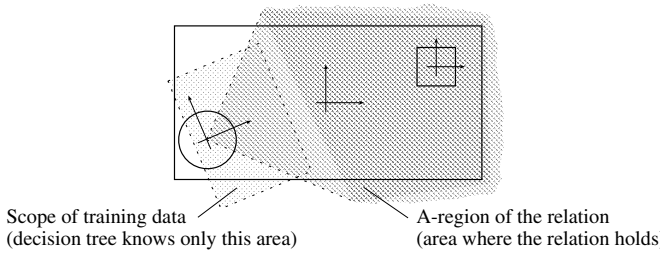


Fig. 9. Scope of training data not adapted to the size of the room

The benefits of our learning approach are to get a new, easier representation of the decision boundary (i.e. the constraints). The new representation contains the solution of the constraints (i.e. the A-regions), and the accuracy of the approximation can be arbitrary high. The drawback, however, is the problem of generating suitable data sets.

4.2 Generating Depictions

In the previous section we transformed the constraints of the spatial relations into a new representation (i.e. the learned classifiers). As it was our aim to solve the spatial constraints, we present now an algorithm (see Fig. 10) for generating depictions, which uses the learned CAL5 decision trees.

As mentioned above, for every needed relation and for every pair of object types, corresponding (sub-)relations must be learned. Since we are interested in those regions in the parameter space where the relation holds, we do some preprocessing. This means, for each leaf node of the decision tree in the tree with class A the admissible intervals for every parameter are being extracted. Note, that each depiction is a point in the configuration space. These points are represented by vectors, which contain all variables of the problem. Recall that our relations are binary. Thus, we get three cases: both objects are 'unknown', one object is already placed or both objects are placed. In the first case, we place one object randomly in the room (lines 09 and 10 in Fig. 10). This leads us to the second case. There we pick a class A leaf of the tree and compute the size, relative position and relative orientation of the other object by assigning values to the remaining variables of this object within the intervals of the chosen leaf (lines 11 and 12). If the collision check in line 13 fails (for both, case one and two), we repeat the procedure up to k times (block between lines 08 and 14). If

[4] `'r'` denotes a `rectangle`, and `'c'` a `circle`

we do not have admissible values after the kth trial (line 15), we suppose that the current relation cannot hold in combination with the others and we reject the depiction generated so far. However, there may exist solutions. Actually we cannot distinguish between the case "no solution" and "disadvantageous values". So if we reject the vector, we have to start again with the first relation. For practical reasons we work instead on a number of object constellations in parallel. This means, we are starting with v empty vectors and apply the algorithm to each vector. Thus, we obtain in each step at most v depictions (valid for the relations processed so far). If we have to drop a vector, we still have $v_n - 1$ other scenes in step n. It is not critical to discard "disadvantageous values" because usually the scenes are underconstrained. In the last case both objects are already placed and we have to check, whether the values of the objects range in the intervals represented by at least one leaf (lines 05 and 06). If not, the relations do not hold, at least for the calculated values.

This procedure is repeated for every relation with the remaining objects, which satisfy the relations processed in the former steps. Finally we obtain up to v depictions according to the given spatial description. In the case, that we have found no depiction, we have to assume that the constraints are unsatisfiable.

Up to now each decision tree represents only the constraints for the particular relations with the two objects of the specified forms. The background knowledge (e.g. the objects must not overlap) was not learned but checked after every step explicitly. In general, it is possible to learn the background knowledge constraints as well and to check them like the other relations.

The parameters v and k depend on the given relations and on the number of relations. They have to be chosen large enough to get a correct answer ("There is no solution." or "We have found at least one.") by some probability. At the same time, one should choose rather small v and k, because by increasing the parameters the calculation time increases, too. So they have to be chosen in relation to the problem to be solved.

As shown in Table 2, the more relations to solve, the higher v has to be. A value of 100 for k seems to be a good choice. The number of trials per valid solution increases exponentially in the number of relations to solve. Not shown, but critical is the processing sequence. Very restrictive relations like `atwall/2` should be solved at the beginning.

Example 3. Supposed we have two relations, `right(cupboard, lamp)` and `atwall(wall1, cupboard)`. Now we fulfill first the `right/2` relation. Therefore the cupboard may be placed somewhere in the room. After that we cannot satisfy `atwall/2`, because the cupboard should be placed near `wall1`, but actually it is already placed in the room. So we would have to increase v, but nevertheless the probability to get a solution is very small.

5 Discussion

In the previous sections we sketched two approaches to spatial reasoning. First, we have shown how to use a system of cooperating constraint solvers, where

Depiction generation algorithm

```
INPUT: number v of initial vectors and number k of trials
       relations r that have to hold
OUTPUT: up to v depictions, where all relations r hold
ALGORITHM:
01  foreach relation r:
02    identify objects and object types by object descriptions
03    load the corresponding (pruned) decision tree
04    foreach vector v:
05      if both objects were placed
06      then check whether relation r holds
07      else
08        repeat up to k times:
09          if both objects are new
10          then place first object randomly in room
11          pick area (random according to weight of leaf)
12          assign values to variables within intervals of leaf
13          check non-overlapping with other objects and walls
14        until check passes
15      if no success
16      then drop vector v
17  show remaining vectors (depictions)
```

Fig. 10. Algorithm for generation depictions using decision trees

the inference of an interval arithmetic solver is supported by further solving methods. In the second approach we applied machine learning in combination with a new algorithm for depiction generation to this kind of problem.

The direct constraint solving approach may at first seem to be evident. Using our system of cooperating solvers the restriction of the solution space is possible as well as to compute particular depictions. The system is comfortable in use and can be extended by new solvers in a simple way. However, the disadvantages of the possible incompleteness of the solvers, and, thus, as well of the overall system, and an increasing time effort by increasing complexity of the spatial scene must be taken into consideration.

Using the learning approach yields decision trees, which are very well interpretable. The approximation of the decision boundaries may be (at least in principle) arbitrarily high. Generating suitable training sets, however, is not trivial. The depiction generation algorithm employs the decision rules for restricting the space of possible solutions. In the limit of generating an infinite number of depictions (i.e. exhaustive search) the algorithm finds every possible solution. Because the processing sequence of the relations is critical, we may find no solution, although there is one. However, the scene descriptions in this problem domain are usually underconstrained, and, thus, it is usually not a problem

Table 2. Some test runs and typical results of our algorithm

Combination of relations	suitable v/k	number v per valid depiction (average)
single relation, e.g. `right(steffi, cupboard)`	100/10	2

two relations, e.g.
`right(steffi, cupboard)`
`front(steffi, fridge)`
	1.000/100	10

three relations, e.g.
`right(steffi, cupboard)`
`left(fridge, lamp)`
`right(cupboard, lamp)`
	1.000/100	61

four relations, e.g.
`right(steffi, cupboard)`
`left(fridge, lamp)`
`right(cupboard, lamp)`
`front(steffi, fridge)`
	1.000/100	375

five relations, similar to above	10.000/100	638

to find an alternative solution by constructing another sequence (i.e. following another path in the problem space).

Comparing the approaches of learning on the one hand and constraint solving on the other hand, at first they seem to be very different. However, they are not: The constraint solving approach takes the given constraints, checks their satisfiability, and tries to compute solutions. The learning approach generates rules for constraint decision using training data. This means, however, that these rules together with the search procedure build as well a constraint solver for the provided constraints; its behavior depends on the kind, amount and complexity of training data.

We think, that the direct solving approach can be useful for a more large-grain scene estimation, while the learning approach can allow (dependent on the provided training data) a more fine-grain consideration and as well looking at exceptions. This yields further research perspectives: Constraint solving could be used for an early exclusion of unsatisfiable scene descriptions. Because of the possible incompleteness of the solvers, it will not in general detect every inconsistency. For a more fine-grain elaboration of the remaining areas the learning approach could yield then better results. A second promising idea is to use constraints and constraint solving to prestructure the rule set for the learning method and to tare particular parameters afterwards using the learning mechanism with selected training data. This may allow to reduce the training cost and increase the speed of the depiction generation.

References

1. The Brandeis Interval Arithmetic Constraint Solver, March 2002. Available from `http://www.cs.brandeis.edu/~tim/`.
2. A. P. Ambler and R. J. Popplestone. Inferring the Positions of Bodies from Specified Spatial Relationships. *Artificial Intelligence*, 6:157–174, 1975.
3. B. Claus, K. Eyferth, C. Gips, R. Hörnig, U. Schmid, S. Wiebrock, and F. Wysotzki. Reference Frames for Spatial Inferences in Text Comprehension. In C. Freksa, C. Habel, and K. F. Wender, editors, *Spatial Cognition*, LNAI 1404. Springer, 1998.
4. P. Geibel, C. Gips, S. Wiebrock, and F. Wysotzki. Learning Spatial Relations with CAL5 and TRITOP. Technical Report 98-7, TU Berlin, 1998.
5. P. Griebel, G. Lehrenfeld, W. Mueller, C. Tahedl, and H. Uhr. Integrating a Constraint Solver into a Real-Time Animation Environment. In *Proc. of the 1996 IEEE Symposium on Visual Languages*, September 1996.
6. H. W. Guesgen. Spatial Reasoning Based on Allen's Temporal Logic. Technical Report TR-89-049, ICSI, Berkeley, Cal., 1989.
7. S. Haridi, S. Janson, J. Montelius, T. Franzen, P. Brand, K. Boortz, B. Danielsson, B. Carlson, T. Keisu, D. Sahlin, and T. Sjöland. Concurrent Constraint Programming at SICS with the Andorra Kernel Language. In P. Kanellakis, J.-L. Lassez, and V. Saraswat, editors, *First Workshop on Principles and Practice of Constraint Programming – PPCP'93*, 1993.
8. Daniel Hernández. *Qualitative Representation of Spatial Knowledge.* LNAI 804. Springer, 1994.
9. T.J. Hickey, M.H. van Emden, and H. Wu. A Unified Framework for Interval Constraints and Interval Arithmetic. In M. Maher and J.-F. Puget, editors, *Principles and Practice of Constraint Programming – CP'98*, LNCS 1520. Springer, 1998.
10. P. Hofstedt. Better Communication for Tighter Cooperation. In *First International Conference on Computational Logic*, LNCS 1861. Springer, 2000.
11. P. Hofstedt, E. Godehardt, and D. Seifert. A Framework for Cooperating Constraint Solvers - A Prototypic Implementation. In E. Monfroy and L. Granvilliers, editors, *Workshop on Cooperative Solvers in Constraint Programming - CoSolv*, 2001.
12. P. N. Johnson-Laird. *Mental Models: Towards a Cognitive Science of Language, Inference and Consciousness.* Cambridge University Press, Cambridge, 1983.
13. T. Jörding and I. Wachsmuth. An Antropomorphic Agent for the Use of Spatial Language. In *Proceedings of ECAI'96-Workshop on Representation and Processing of Spatial Expressions*, pages 41–53, 1996.
14. G. Nakhaeizadeh and C. C. Taylor, editors. *Machine Learning and Statistics - The Interface.* Wiley, 1997.
15. T. Röfer. Routemark-based Navigation of a Wheelchair. In *Third ECPD International Conference on Advanced Robotics, Intelligent Automation and Active Systems*, Bremen, 1997.
16. S. Wiebrock, L. Wittenburg, U. Schmid, and F. Wysotzki. Inference and Visualization of Spatial Relations. In C. Freksa, W. Brauer, C. Habel, and K. Wender, editors, *Spatial Cognition II*, LNAI 1849. Springer, 2000.
17. S. Wiebrock and F. Wysotzki. Lernen von räumlichen Relationen mit CAL5 und DIPOL. Technical Report 99-17, TU Berlin, 1999.

From Simulated Dialogues to Interactive Performances with Virtual Actors

Elisabeth André

University of Augsburg

Abstract. In my talk, I will argue in favor of a shift from applications with single presentation agents towards flexible performances given by a team of characters as a new presentation style. Infotainment and edutainment transmissions on TV as well as advertisement clips are examples that demonstrate how information can be conveyed in an appealing manner by multiple presenters with complementary characters and role castings. However, our approach distinguishes from conventional TV presentations by at least two features: adaptivity and interactivity. I will illustrate the approach by means of various academic and industrial projects we conduced at DFKI GmbH and at Augsburg University. In the first group of systems, the attribute "flexible" refers to the system's ability to adapt a presentation to the needs and preferences of a particular user. In the second group of systems, flexibility additionally refers to the user's option of actively participating in a computer-based performance and influencing the behavior of the involved characters at runtime. While a plan-based approach has proven appropriate in both versions to automatically control the behavior of the agents, the second group of systems calls for highly reactive and distributed behavior planning.

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, p. 317, 2002.

Time, Knowledge, and Cooperation: Alternating-Time Temporal Epistemic Logic and Its Applications

Michael Wooldridge

University of Liverpool

Abstract. Branching-time temporal logics have proved to be an extraordinarily successful tool in the formal specification and verification of distributed systems. Much of this recent success stems from the tractability of the model checking problem for the branching time logic CTL. Several successful verification tools (of which SMV is the best known) have been implemented that allow designers to verify that systems satisfy requirements expressed in CTL. Recently, CTL was generalized by Alur, Henzinger, and Kupferman in a logic known as "Alternating-time Temporal Logic" (ATL). The key insight in ATL is that the path quantifiers of CTL could be replaced by "cooperation modalities", of the form $\ll G \gg$, where G is a set of agents. The intended interpretation of an ATL formula $\ll G \gg \Phi$ is that the agents G can cooperate to ensure that Φ holds (equivalently, that G have a winning strategy for Φ). It turns out that the resulting logic very naturally generalizes and extends CTL. In this talk, I will discuss extensions to ATL with *knowledge modalities*, of the kind made popular by the work of Fagin, Halpern, Moses, and Vardi. Combining these knowledge modalities with ATL, it becomes possible to express such properties as "group G can cooperate to bring about Φ iff it is common knowledge in G that Φ". The resulting logic – Alternating-time Temporal Epistemic Logic (ATEL) – has a range of applications, which will be discussed in the talk. In addition, I will relate some preliminary experiments with ATEL model checking, which shares the tractability property of its ancestor CTL. (This talk reports joint work with Wiebe van der Hoek.)

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, p. 318, 2002.

Semantic Web Enabled Web Services

Dieter Fensel

Vrije Universiteit Amsterdam

Abstract. Currently, computers are changing from single, isolated devices into entry points to a worldwide network of information exchange and business transactions called the World Wide Web. However, the easy information access based on the success of the web has made it increasingly difficult to find, present, and maintain the information required by a wide variety of users. In response to this problem, many new research initiatives and commercial enterprises have been set up to enrich available information with machine-understandable semantics. This semantic web will provide intelligent access to heterogeneous, distributed information, enabling software products to mediate between user needs and the information sources available. Web Services tackle with an orthogonal limitation of the current web. It is mainly a collection of information but does not yet provide support in processing this information, i.e., in using the computer as a computational device. Recent efforts around UDDI, WSDL, and SOAP try to lift the web to a new level of service. Software programs can be accessed and executed via the web. However, all these service descriptions are based on semi-formal natural language descriptions. Therefore, the human programmer need be kept in the loop and scalability as well as economy of web services are limited. Bringing them to their full potential requires their combination with semantic web technology. It will provide mechanization in service identification, configuration, comparison, and combination. Semantic Web enabled Web Services have the potential to change our life in a much higher degree as the current web already did.

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, p. 319, 2002.

DFG Priority Program RoboCup (SPP-1125) Cooperating Teams of Mobile Robots in Dynamic Environments

Thomas Christaller

Fraunhofer AIS

The priority program 1125 of the Deutsche Forschungsgemeinschaft funds research on methods, components and tools in the domain of cooperating teams of mobile robots operating in dynamic environments. An important goal of the six-year program is to foster synergistic collaboration of research groups working especially in the context of RoboCup. The first two-year phase started in June 2001 with 14 projects. A list of all funded projects can be found at http://www.ais.fraunhofer.de/dfg-robocup. Each project contributes to at least one of the following working groups.

WorkingGgroup "System Integration" (AG1)
This working group concentrates on methods and tools required to integrate hardware and software components of a mobile robot team to a whole system. Aspects are descriptions of and interfaces to sensors, actuators, robot control programs and middleware for inter-robot communication. One goal of the first phase is the definition of a simulation framework which allows to integrate and exchange components and results from different projects.

Working Group "Learning" (AG2)
All projects working on machine learning (reinforcement learning, Kohonen, neuro-evolutive) are cooperating in this working group. One goal is to apply the results from machine learning achieved in simulated virtual environments to real robots. This includes basic skill learning or learning of team behavior. In order to compare different learning approaches, benchmarks will be defined.

Working Group "Architecture" (AG3)
This working group deals with models and representations relevant for mobile robot teams. This includes local and global world models derived from sensoric input for navigation and self-localisation. In the context of multi-agent systems, research topics are detection of tactic and strategy of a single robot or a team of robots, cooperation in a team of robots, detection of plans as well as development and exchange of distributed plans.

`http://www.ais.fraunhofer.de/dfg-robocup`

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, p. 323, 2002.

DFG Priority Program (SPP-1083) Intelligent Agents and Realistic Commercial Application Scenarios

Stefan Kirn

Ilmenau University of Technology

Besides the development of the object-oriented paradigm within software engineering, the area of intelligent software agents was established as a new subdiscipline of artificial intelligence in the late 1970s. In a sense, it can also be understood as an extension of the object-oriented paradigm by introducing "intelligent objects" into this field. Intentional models define agents as software systems which autonomously adjust their behavior in accordance with dynamically changing goals. With this definition, artificial intelligence has taken a step from a technological level to research areas which lie at the heart of economic theories. For example, the concept of limited rationality of software agents on the one hand and the model of the so-called "homo oeconomicus" ground on the same roots, and there are relationships between the manager/contractor model in negotiation-based coordination and the approach of principal/agent theory in economics, or between models of cooperation processes in multi-agent systems and transaction-cost theoretic approaches in the New Institution Economics.

On this basis researchers from management science, information systems, and computer science are collaborating in order to advance the state of the art in intelligent software agents so that agent technologies for large systems in realistic commercial application scenarios can be developed and tested. With that one goal is to find and investigate a new approach to the development of application systems and to an improved adaptivity of companies with respect to dynamic market processes. In particular, this approach should more closely meet the requirements of the networking and the dynamics of worldwide distributed business processes than existing proposals.

The program focuses on the logistics domain. For one, this business cross-section function is particularly affected by the networking phenomenon. For another, research in Germany on agent-based application scenarios has reached a leading position in recent years. Information logistics and material-goods logistics as two typical and important business application and research areas are to be investigated in the manufacturing industry, where one can already build on a large body of experience, and the health care domain. Especially the latter is characterized by dynamic relationships regarding the planning and execution of service processes and is hence a perfect reference domain for large agent- based systems. Another focus is the so far largely unexplored business- and market-related impact of agent technology. The transdisciplinary goals of this priority program will have fruitful repercussions on business theories and reference models, and it will enable the participating research groups to play a leading role in standardization efforts for agent technologies.

`http://www.wirtschaft.tu-ilmenau.de/wi/wi2/SPP-Agenten`

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, p. 324, 2002.

DFG Priority Program (SPP-1077) Socionics – Investigating and Modelling Artificial Societies

Thomas Malsch

Hamburg-Harburg University of Technology

Classical AI has a long history of building computer systems according to models of the human brain. In contrast, Distributed Artificial Intelligence (DAI) assumes that intelligence is inherently social, emerging from the interactions of multiple autonomous and dispersed entities. From the very beginning of their work, researchers in DAI have used metaphors of the social world as a source of inspiration. However, this occupation was based on a rather intuitive understanding of how human societies are organised – despite appeals to systematically lay "firm social foundations for DAI research" (Les Gasser).

The research programme of *Socionics* addresses exactly this demand. It brings together researchers from the field of DAI with sociologists in an interdisciplinary attempt to answer the fundamental question: How to exploit models from the social world for the development of intelligent computer technologies? Work in *Socionics* demands from sociologists to precisely specify the conceptual apparatus with which they observe and explain the unique features of human society (flexibility, combined with ultra-robustness). From computer scientists, it demands that they design large-scale systems, which are in the end non-deterministic. To both, it offers a fascinating potential to stimulate interdisciplinary research and to develop innovative applications.

Research in *Socionics* claims to address three aspects of computational social intelligence:

- It strives to develop new techniques and methods in DAI by sociologically enhancing the construction of large-scale multiagent systems beyond the scope of contemporary approaches. The reference here is clearly "computational".
- The use of computer models in sociological theorizing marks the "sociological reference", claiming to gain new perspectives on classical problems of sociology, as well as new methodologies to reason about them (e.g. agent-based social simulation).
- Finally, since they are directly involved in the development of new technologies with a "praxis reference", researchers in *Socionics* also gain deep insights into hybrid artificial societies, in which human beings and technical agents interact on the same level.

The research programme is organized in 12 individual projects, each run jointly by "tandems" of computer scientists and sociologists. The projects have successfully finished the first phase, which was concerned with establishing the interdisciplinary agenda and with achieving a multi-paradigm view on sociologically

M. Jarke et al. (Eds.): KI 2002, LNAI 2479, pp. 325–326, 2002.

enhanced DAI. The current second phase focuses on the problem of scalability, a central problem for DAI, with open questions not only with computational, but also sociological and praxis reference. This problem is ideally suited to demonstrate the potential of the *Socionics* research programme to produce innovative solutions.

For more information, please look at the programme's web site or contact the programme's coordinator, Prof. Dr. Thomas Malsch.

`http://www.tu-harburg.de/tbg/SPP/Start_SPP.html`

Author Index

Alvarado, P., 253
André, E., 317
Arens, M., 268

Baader, F., 99
Bonzon, P., 33
Borth, M., 82
Bui, T.D., 129
Burghardt, J., 222

Christaller, T., 323
Ciortuz, L., 3

Dixon, C., 235
Dörfler, P., 253

Fensel, D., 319
Fernández Gago, M.C., 235
Fisher, M., 235

Geibel, P., 186
Gips, C., 299
Guetova, M., 67

Heylen, D., 129
Hitzler, P., 205
Hofstedt, P., 299
Hölldobler, S., 67

Jain, B.J., 163

Kirn, S., 324
Kraiss, K.-F., 253
Krüger, T., 253

Malsch, T., 325
Murray, J., 283
Müller, M., 144

Nagel, H.-H., 268
Ney, H., 18
Nijholt, A., 129

Obst, O., 283
Och, F.J., 18

Poel, M., 129

Schädler, K, 186
Schmid, U., 144
Schwind, M., 51
Skabar, A., 174
Störr, H.-P., 67
Stolzenburg, F., 283
Stuckenschmidt, H., 114

Turhan, A.-Y., 99

Wendt, M., 205
Wendt, O., 51
Wickel, J., 253
Wooldridge, M., 318
Wysotzki, F., 144, 163, 186, 299

Zens, R., 18