Universität Karlsruhe

Institut für Logik, Komplexität und Deduktionssysteme

Vorlesungsskriptum

Automatisches Beweisen

Sommersemester 2001

überarbeitet von

Bernhard Beckert

beckert@ira.uka.de
i12www.ira.uka.de/~beckert

erstellt von

Reiner Hähnle

reiner@cs.chalmers.se
www.cs.chalmers.se/~reiner

Inhaltsverzeichnis

Vorwort

Zur Auflage von 1997

Dieses Skriptum ist aus den Folien und Notizen für die Vorlesung „Automatisches Beweisen“ entstanden, die ich zuerst im Sommersemester 1994 gehalten habe. Im darauffolgenden Jahr wurde die Vorlesung unverändert von Prof. Peter Schmitt gehalten. Im Sommersemester 1997 las ich die Vorlesung wieder selbst und nahm dies zum Anlaß, eine korrigierte und erweiterte Fassung zu erstellen, die hiermit vorliegt.

In der Vorlesung wollte ich einige Lerninhalte vermitteln, die meines Erachtens für eine zeitgemäße Einführung in das Thema „Automatisches Beweisen“ unerläßlich sind:

- Vorstellung und Diskussion aller wichtigen Kalkülklassen und Inferenzprozeduren im Automatischen Beweisen.
- Eingehen auf die Zusammenhänge zwischen den Kalkülen und auf ihre Grenzen.
- Einordnung der besprochenen Kalküle bezüglich Anwendungen und möglicher Erweiterungen.

Es existiert zwar eine Vielzahl von Büchern zum Thema der Vorlesung, doch bei näherem Hinsehen erwies sich keines von ihnen als Begleittext zu einer Vorlesung mit dem Inhalt, wie ich ihn geplant hatte, als geeignet. Die meisten Bücher beschäftigen sich nur mit relativen speziellen Klassen von Kalkülen und verwenden eine jeweils unterschiedliche Terminologie für diese. Andere Werke streuen zu sehr in die Breite und sind daher höchstens als Orientierung zu empfehlen. Einige Texte sind schlicht veraltet. Dazu kommt, daß insbesondere neuere Entwicklungen bei Tableaukalkülen bislang nur in Zeitschriftenartikeln und Doktorarbeiten zugänglich sind. All dies bewog mich dazu, den Versuch zu machen, aus dem vorhandenen Material zumindest das Skelett eines Buchs zu synthetisieren.

Naturgemäß ist der vorliegende Text an vielen Stellen noch unvollständig. Bei vielen Themen, die in der Vorlesung mündlich erläutert wurden, mußte auf die Ausformulierung verzichtet werden. Schließlich erforderte der Rahmen einer Vorlesung im Umfang von drei SWS eine gewisse Reduktion des Materials und bedingte notwendigerweise die Vernachlässigung einiger Themen zugunsten anderer. Inbesondere die Behandlung von Gleichheit sollte ausführlicher sein als sie es gegenwärtig ist. Um aber die Produktion eines brauchbaren Textes nicht *ad infinitum* hinauszuzögern, habe ich mich dazu entschlossen, es bei dem in der Vorlesung behandelten Stoff zu belassen. Auch so sind einige Abschnitte recht spröde geworden und vertrügen mehr Erklärungen; ich habe versucht, diesem Mißstand mit vielen gezielten Literaturhinweisen etwas abzuhelfen, so daß der Text mit Vorkenntnissen, wie sie etwa im Rahmen der Vorlesung „Formale Systeme“ oder durch die Lektüre eines Buchs wie das von Schöning (1992) erworben werden, erarbeitet werden kann. In Kapitel 2 sind die benötigten grundlegenden Resultate und Definitionen aus der Logik kurz zusammengefaßt; eine richtiggehende Einführung in diese kann jenes Kapitel freilich nicht ersetzen.

Der Rest des Textes folgt einem einfachen Schema: die besprochenen Kalküle werden grob in zwei Familien eingeteilt: Tableaumethoden und Resolutionsmethoden. Beide werden in allgemeiner Weise in Kapitel 3 bzw. in Kapitel 7 eingeführt und in den jeweils unmittelbar darauffolgenden Kapiteln verfeinert. In Kapitel 10 werden praktische und theoretische Eigenschaften beider Kalkülfamilien verglichen, in Kapitel 11 werden einige konkrete Implementierungen vorgestellt, schließlich werden in Kapitel 12 einige Themen skizziert, die im Haupttext nicht angesprochen wurden, obwohl sie durchaus zum Gebiet „Automatisches Beweisen“ gehören. Dort werden auch Hinweise auf weitere Lehrveranstaltungen an der Universitat Karlsruhe mit Bezug zu dieser Vorlesung gegeben.

Ich möchte mich an dieser Stelle herzlich bedanken bei: der Fakultät für Informatik an der Universität Karlsruhe, die mir den Lehrauftrag für die Vorlesung „Automatisches Beweisen“ erteilt hat; bei den Studierenden, die in der Vorlesung waren, und denen ich viele Anregungen und Korrekturen zu verdanken habe; sowie bei den Herren Dr. David Basin, Bernhard Beckert und Dr. Joachim Posegga, die frühere Versionen dieses Skriptums kritisch gelesen haben.

Karlsruhe, Juli 1997

Reiner Hähnle

Vorwort zur Auflage von 1999

Im Sommersemester 1999 las ich die Vorlesung „Automatisches Beweisen“ zum ersten mal; ich verwendete dabei das von Dr. Reiner Hähnle erstellte Skriptum. Im wesentlichen blieb der Inhalt der Vorlesung unverändert – mit Ausnahme eines neues Kapitels über Theorieschließen.

Die vorliegende, überarbeitete Version des Skriptums habe ich im Anschluß an die Vorlesung erstellt. Ich habe einige kleinere Änderungen und Fehlerkorrekturen vorgenommen und das neue Kapitel 9 (Theorieschließen) hinzugefügt.

Ich danke der Fakultät für Informatik an der Universität Karlsruhe für die Erteilung des Lehrauftrags, Herrn Dr. Reiner Hähnle dafür, daß er mir sein Lehrmaterial zur Verfügung gestellt hat, und meinen Hörern für ihre engagierte Teilnahme an der Vorlesung.

Karlsruhe, August 1999

Bernhard Beckert

Vorwort zur Auflage von 2001

Nachdem ich die Vorlesung „Automatisches Beweisen“ auch in den Sommersemestern 2000 und 2001 wieder gelesen hatte, habe ich einige Fehler in diesem Skriptum korrigiert.

Weitere Informationen zur Vorlesung finden sich unter der URL

`i12www.ira.uka.de/~beckert/automatisches-beweisen-SS99/`

Karlsruhe, August 2001

Bernhard Beckert

Kapitel 1

Einführung

Den Anfang des Gebiets „Automatisches Beweisen“ kann man auf die Mitte der fünfziger Jahre datieren. Damals wurden erste Versuche unternommen, mathematische Resultate automatisch mit einem Computerprogramm herzuleiten. Daß es prinzipiell möglich ist, mathematische Wahrheiten formal zu modellieren und mittels einer beschränkten Anzahl von Axiomen und Regeln formal zu beweisen, wußte man seit den grundlegenden Untersuchungen zur Axiomatisierbarkeit der Logik und Mathematik in der ersten Hälfte unseres Jahrhunderts.

Wie in Abbildung 1.1 zu sehen, werden mathematische Probleme formalisiert, das heißt in einer geeigneten formalen Sprache dargestellt.[1] Wir denken dabei vor allem an logische Syntax, anderes ist aber genauso möglich. Formalen syntaktischen Objekten (in der Logik sind dies im allgemeinen Aussagen) wird mittels einer *Wahrheitsfunktion*, die jeder Aussage einen *Wahrheitswert* zuordnet, eine *Semantik* gegeben. Mittels der Semantik kann die Adäquatheit der Formalisierung überprüft werden. Den Gesamtprozeß nennt man *Modellierung*. Syntax und Semantik zusammen definieren eine *Logik*.[2]

Ist der Modellierungsprozeß abgeschlossen, kann von der Gültigkeit einer formalen Aussage – die *Formalisierungstreue* der Modellierung vorausgesetzt – auf die Gültigkeit des entsprechenden realen Sachverhalts geschlossen werden.

Für viele Logiken kann die Wahrheitsfunktion durch eine syntaktisch definierte **Ableitungsrelation** zwischen Mengen von Aussagen charakterisiert oder doch zumindest sinnvoll approximiert werden. Kann jede gültige Aussage abgeleitet werden, spricht man von *Vollständigkeit*, ist umgekehrt jede abgeleitete Aussage wahr, spricht man von Korrektheit. Letzteres stellt üblicherweise eine

[1] Man beachte, daß mathematische Arbeiten und Bücher normalerweise nicht in einer formalen Sprache verfaßt sind, obwohl sie durchaus Elemente formaler Notation enthalten. Diese ist jedoch *ad hoc* und nicht etwa durch eine Chomsky-Grammatik gegeben.

[2] Manche Logiker bevorzugen eine Logik über einen Beweiskalkül zu definieren, der bei uns die sekundäre Rolle spielt. Allerdings haben wir völlig offengelassen, wie eine Semantik beschaffen sein soll, so daß dieser Zugang unserem nicht unbedingt widerspricht.

unverzichtbare Minimalforderung dar.

Der Prozeß des Findens einer Ableitung heißt *Deduktion*, die der Menge Ableitungsregeln heißt (Logik-)*Kalkül*.

Wer jedoch jemals einen formalen Beweis auch nur eines einfachen mathematischen Sachverhalts in einem, sagen wir, Hilbertkalkül gemacht hat, dem ist klar, daß solche formalen Beweise außerhalb der mathematischen Logik niemals eine Rolle spielen werden, solange sie von Menschen erbracht werden müssen.

Mit dem Aufkommen leistungsfähiger Computer hat sich die Situation völlig verändert, denn prinzipiell kann man Computerprogramme schreiben, die den Menschen bei der Ableitungsfindung, also bei der Deduktion unterstützen. Geschieht dies sogar vollautomatisch, spricht man von *Automatischer Deduktion* oder von *Automatischem Beweisen*. Dies ist das Thema der Vorlesung.

Wie in Abbildung 1.1 angedeutet, sieht man im Automatischen Beweisen meist von der Herkunft des zu lösenden Problems ab. Die verwendeten Techniken sind im allgemeinen anwendungsunabhängig. Dies hat mit der Geschichte des Gebiets zu tun:

Obwohl die Ursprünge und Grundlagen des Automatischen Beweisen in der formalen Logik liegen, wird das Gebiet im allgemeinen unter die Disziplin „künstliche Intelligenz“ (KI) subsumiert. In den sechziger Jahren nämlich hegte man die Hoffnung, intelligente Leistungen verschiedenster Art durch regelbasierte Repräsentationen charakterisieren zu können und somit „intelligentes“ Verhalten durch einen hinreichend mächtigen Inferenzmechanismus, den sogenannten *General Problem Solver*, erzielen zu können. Dies wird heute nicht mehr ganz so gesehen und dementsprechend ist Automatisches Beweisen auch nicht mehr im KI „Mainstream“.

Ein schwieriges Problem für typische Logikanwendungen in der KI ist schon die adäquate Formalisierung, in Abbildung 1.1 durch die helle Farbe des typischen Vertreters *natürlichsprachlicher Diskurs* gekennzeichnet. Demgegenüber ist die Formalisierung der Problemstellung *Programmverifikation* vergleichsweise einfach, da Programme schon syntaktisch formale Objekte sind.

Den potentiellen Anwendungen des Automatisches Beweisens in der Kognitionswissenschaft folgten im Laufe der Jahre mehrere andere, die mehr Erfolg hatten. Der Erfolg war sogar so groß, daß dies zum Teil eigenständige Gebiete wurden oder in solche einflossen, zum Beispiel Algebrasysteme, logisches Programmieren (Prolog), Expertensysteme. Obwohl alle drei zumindest teilweise als Spin-offs des Automatischen Beweisens betrachtet werden können, werden sie heute nicht mehr dazu gezählt.

Eine zentrale Technik in der künstlichen Intelligenz freilich findet sich im automatischen Beweisen ebenfalls an zentraler Stelle wieder: die Repräsentation und Erschließung extrem großer Suchräume.

Hier hat sich in den letzten Jahren gezeigt, daß anwendungsunabhängige Techniken einfach nicht stark genung sind, um vielen praxisrelevanten Problemen gerecht zu werden. Heuristisches Wissen zur Organisation großer Suchräu-

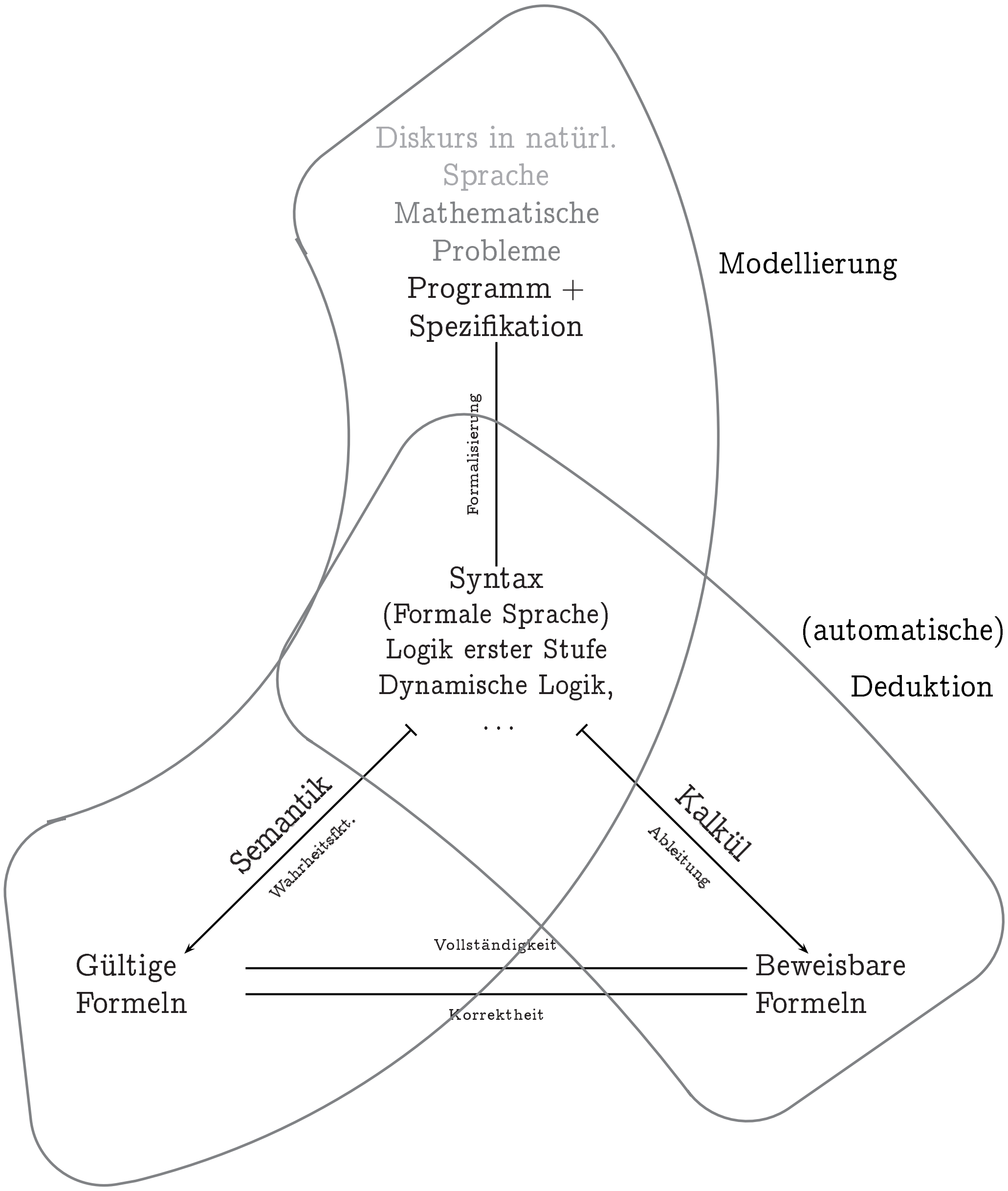


Abb. 1.1: Einordnung der automatischen Deduktion.

me ist unabdingbar, muß jedoch mit Techniken des Automatischen Beweisens kompatibel sein, so daß die Korrekheit und Vollständikeit der Deduktionsalgorithmen gewahrt bleibt. Erst in jüngerer Zeit hat sich die Forschung dieses Problems angenommen. Wir werden in Abschnitt 5.4 ein Beispiel kennenlernen.

Wie wir sehen werden, gibt es zwei große Familien von Beweiskalkülen, die sich hinsichtlich der Organisation der Beweissuche grundlegend unterscheiden. Dies wird in der Aufteilung der Vorlesung in Tableau- und Resolutionsmethoden widergespiegelt. Es entspricht dem Ziel der Vorlesung, dem/der Hörer/in nicht nur die wichtigsten Methoden des Automatischen Beweisens in einem über die „Formale Systeme“ Vorlesung hinausgehenden Detaillierungsgrad zu vermitteln, sondern ihm oder ihr einen Überblick über das Gebiet zu geben, der die Einordnung neuerer Entwicklungen oder spezieller Ergebnisse, die hier nicht besprochen werden, leichter macht.

Die Auswahl eines Kalküls alleine ist jedoch noch nicht entscheidend für die Performanz eines Beweisers. Ebenso wichtig, jedoch für theoretische Untersuchungen weniger interessant, sind die Vorverarbeitung bzw. Aufbereitung des Problems (z.B. Berechnung von Normalformen oder potentieller Konnektionen) sowie Implementierungstechniken (z.B. abstrakte Maschinen oder *finite discrimination trees*). Obwohl der Schwerpunkt der Vorlesung auf der Präsentation verschiedener Kalküle liegt, werden wir an geeigneter Stelle auch Vorverarbeitungs- und Implementierungstechniken ansprechen.

Es ist nicht möglich, in einer dreistündigen Vorlesung alle Teilgebiete des Automatischen Beweisens im weiteren Sinne anzusprechen. Wir haben drei wesentliche Einschränkungen gemacht: zum einen sollte die Vorlesung eigentlich „Vollautomatisches Beweisen“ heißen, denn das schnell wachsende Teilgebiet „taktisches/interaktives Theorembeweisen“ wird nicht berücksichtigt. Auch gehen wir auf das Automatische Beweisen in nichtklassischen Logiken nicht näher ein. Im letzten Kapitel des Skriptums befindet sich eine Liste der nicht besprochenen Themen mit Literaturhinweisen bzw. Hinweisen auf dafür relevante Lehrveranstaltungen an unserer Fakultät.

Geschichtlicher Abriß des Gebiets „Automatisches Beweisen“ Im folgenden sind in tabellarischer Form einige Meilensteine angegeben.

vor 1950	Beweistheoretische Vorarbeiten von Skolem, Herbrand, Gentzen, Schütte
1954	Erster maschinengenerierter Beweis (Davis)
1955ff	Semantische Tableaus (Beth, Hintikka)
1957	Erster maschinengenerierter Beweis in Logikkalkül (Newell & Simon)
1957	*Lazy substitution* durch freie (*dummy*) Variablen (Kanger, Prawitz)
1958	Erster Beweiser für Prädikatenlogik (Prawitz)
1959	Weitere Beweiser (Gilmore, Wang)

1960	Davis-Putnam Verfahren
1963	Unifikation (J. A. Robinson)
1963ff	Resolution (J. A. Robinson); inverse Methode (Maslov)
1963ff	Moderne Tableaus (Smullyan, Lis), ohne Unifikation
1968	Modellelimination (Loveland), mit Unifikation
1971ff	Konnektionsmethode (Bibel), Matings (Andrews), mit Unifikation
1970ff	PROLOG (Colmerauer, Kowalski)
	Verfeinerungen von Resolution, z.B.
	SL-Resolution (Kowalski & Kuehner), cg-Resolution (Kowalski)
1985ff	Beweisen in nichtklassischen Logiken
	Renaissance von Tableaumethoden
1987	Tableaus mit Unifikation

Ausführliche Informationen über die Geschichte des Automatischen Beweisens sind in mehreren, durchweg lesenswerten Artikeln zu finden. Am ausführlichsten ist Mackenzie [1995], etwas kompakter ist Loveland [1984]; die frühen Jahre werden von einem Pionier des Gebiets in Davis [1983] geschildert; speziell zu Tableauverfahren, vor allem den frühen Jahren, siehe Anellis [1990, 1991], zu Resolution den sehr guten Artikel ihres Erfinders J. A. Robinson (1992).

Verwendete Literatur Die grundlegenden Resultate und Definitionen wurden soweit als möglich der Vorlesung „Formale Systeme“ entnommen – siehe das Skriptum von Peter Schmitt (1992a). Der Teil über Tableaukalküle gründet sich unter anderem auf Fitting [1996]; Bibel and Eder [1992]; Letz [1993]; Hähnle [1994]; Beckert and Hähnle [1998]. Der Teil über Resolution folgt in etwa der Darstellung von Leitsch (1997). Für die Vorbereitung der gesamten Vorlesung ist außerdem die von Siekmann & Wrightson herausgegebene Sammlung (1983b), (1983c) klassischer Papiere zum automatischen Beweisen von unschätzbarem Wert gewesen.

Kapitel 2

Logische Grundlagen

In den folgenden Abschnitten stellen wir die aus der Vorlesung „Formale Systeme" benötigten Definitionen und Resultate zusammen. Dabei halten wir uns weitestgehend an die dort eingeführte Notation.

2.1 Aussagenlogik (AL)

Definition 2.1 $\{(,),\underline{true},\underline{false},\neg,\wedge,\vee,\rightarrow,\leftrightarrow\}$ *sei die Menge der* Sonderzeichen *sowie* $\Sigma = \{p_0, p_1, \ldots\}$ *eine* aussagenlogische Signatur*. Dann ist die Menge* $For0_\Sigma$ *der AL-Formeln über* Σ *definiert als die kleinste Menge mit folgenden Eigenschaften:*

1. $\underline{true}, \underline{false} \in For0_\Sigma$, $\Sigma \subseteq For0_\Sigma$.
2. *Wenn* $A, B \in For0_\Sigma$, *dann sind auch* $\neg A$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$ *und* $(A \leftrightarrow B)$ *in* $For0_\Sigma$.

Die Zeichen $\neg,\wedge,\vee,\rightarrow,\leftrightarrow$ *heißen* (aussagen-)logische Operatoren *oder auch* Konnektive *($\underline{true}$ und $\underline{false}$ können als nullstellige Operatoren bzw. Konnektive aufgefaßt werden). Formeln der Gestalt* $\neg p$, p *mit* $p \in \Sigma$ *heißen* Literal*. AL-Variablen werden oft auch als* Atom *bezeichnet.* ∎

Es gelten die üblichen Regeln für das Weglassen von Klammern falls die Präzedenz $\neg > \wedge > \vee > \rightarrow > \leftrightarrow$ der logischen Operatoren zugrundegelegt wird. Zum Beispiel steht $(p \rightarrow q) \wedge (q \rightarrow r) \rightarrow (p \rightarrow r)$ für die Formel $((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r)$.

Definition 2.2 *Eine* Unterformel *einer Formel* $A \in For0_\Sigma$ *ist ein zusammenhängendes Teilwort* B *von* A*, das selbst Formel ist. Falls* $B \neq A$*, ist* B echte Unterformel*; falls* $C = B$ *oder* $C = \neg B$ *ist* C schwache Unterformel *von* A*. Ist schließlich* B *echte Unterformel von* A *und gibt es keine echte Unterformel* C *von* A*, so daß* B *echte Unterformel von* C *ist, heißt* B direkte Unterformel*.* ∎

α	α_1	α_2
$A \wedge B$	A	B
$\neg(A \vee B)$	$\neg A$	$\neg B$
$\neg(A \rightarrow B)$	A	$\neg B$
$\neg\neg A$	A	A

β	β_1	β_2
$\neg(A \wedge B)$	$\neg A$	$\neg B$
$A \vee B$	A	B
$A \rightarrow B$	$\neg A$	B

γ	$\gamma_1(x)$
$\forall x A(x)$	$A(x)$
$\neg \exists x A(x)$	$\neg A(x)$

δ	$\delta_1(x)$
$\neg \forall x A(x)$	$\neg A(x)$
$\exists x A(x)$	$A(x)$

Tabelle 2.1: Zuordnungsregeln von Formeln zu Unterformeln für verschiedene Formelklassen.

Beispiel 2.3 Sei $A = (p \rightarrow q) \rightarrow \neg q$. Die Unterformeln von A sind: A, $p \rightarrow q$, p, $\neg q$, q, alle außer A sind echt; schwache Unterformeln sind außerdem noch: $\neg A$, $\neg(p \rightarrow q)$, $\neg p$; die direkten Unterformeln von A sind: $p \rightarrow q$ und $\neg q$. ∎

Definition 2.4 *Seien* $A, B \in For_\Sigma$. *Die* Polarität $pol : For_\Sigma \times For_\Sigma \rightarrow \{+, -, \pm\}$ *des Auftretens einer Unterformel* B *in einer Formel* A *ist rekursiv wie folgt definiert:*

1. $pol(A, A) = +$.
2. $pol(A, \neg A) = -$.
3. $pol(B, A) = pol(B, s(A))$, *falls* A *in einer linken Spalte in Tabelle 2.1 auftaucht,* $s(A)$ *eine zugehörige direkte Unterformel ist und das Auftreten von* B *in* A *in* $s(A)$ *ist.*
4. $pol(B, A_1 \leftrightarrow A_2) = \pm$ *falls* B *in* $A_1 \leftrightarrow A_2$ *auftritt.*

∎

Definition 2.5 *Die Operatoren* $\{\neg, \wedge, \vee, \rightarrow\}$ *heißen* primäre logische Operatoren. *Wir sagen, daß Formeln der Gestalt* $\neg\neg A$, $A \wedge B$, $\neg(A \vee B)$ *oder* $\neg(A \rightarrow B)$ *vom* Typ α *oder* konjunktiv *sind, während Formeln der Gestalt* $\neg(A \wedge B)$, $A \vee B$, $A \rightarrow B$ *vom* Typ β *oder* disjunktiv *sind. In der oberen Hälfte von Tabelle 2.1 sind Zuordnungsregeln angegeben, die jedem aus primären Operatoren aufgebauten Nichtliteral zwei schwache, direkte Unterformeln zuordnen.* ∎

Diese sog. **unifizierende** oder *uniforme Notation* geht auf Smullyan (1995) zurück. Sie vereinfacht die Beschreibung und Beweisführung in Kalkülen für volle Logiksyntax ungemein, da die meisten Definitionen und Beweise auf struktureller Induktion über den Formelaufbau beruhen. Uniforme Notation reduziert die Anzahl der Fallunterscheidungen drastisch.

Primäre Operatoren sind gerade diejenigen, welche sich mit genau einer Konjunktion bzw. einer Disjunktion ausdrücken lassen. Es gibt noch zwei weitere primäre Operatoren, nämlich $A \leftarrow B$, $\neg(A \leftarrow B)$, die wir nicht verwenden.

Nicht primär ist $\leftrightarrow$ und $\not\leftrightarrow$; letzteres ist auch als *exklusives Oder* bekannt.

Man beachte, daß $A \leftrightarrow B$ gdw $A \rightarrow B$ und $B \rightarrow A$ bzw. gdw $A \wedge B$ oder $\neg A \wedge \neg B$. Jedoch ist $A \leftrightarrow B$ nicht als einfache Konjunktion oder Disjunktion auszudrücken. Die unvermeidbare Verdoppelung der Unterformeln bei der Auflösung von $\leftrightarrow$ und $\not\leftrightarrow$ kann zu einem massives Problem werden.

Die uniforme Notation erlaubt auch, in einfacher Weise die *Polarität* einer Unterformel zu definieren:

Definition 2.6 *Die* Polarität *des Auftretens*[1] *einer Unterformel* B *in* A *ist* $pol : For_\Sigma \times For_\Sigma \rightarrow \{+, -, \pm\}$ *mit:*

1. $pol(A, A) = +$
2. $pol(A, \neg A) = -$
3. $pol(B, A) = pol(B, s(A))$, *falls* $s(A)$ *eine der durch uniforme Notation* A *zugeordneten Unterformeln ist und das Auftreten von* B *in* $s(A)$ *ist.*
4. $pol(B, A_1 \leftrightarrow A_2) = \pm$

■

Definition 2.7 *Relativ zu einer Signatur* Σ *wird eine* aussagenlogische Interpretation $I : \Sigma \rightarrow \{W, F\}$ *angegeben. Die Elemente von* $\{W, F\}$ *heißen* Wahrheitswerte*. Jede Interpretation* I *induziert eine* Auswertungsfunktion *auf Formeln* $val_I : For0_\Sigma \rightarrow \{W, F\}$ *vermöge der Vorschrift:*

$$
\begin{array}{ll}
val_I(\underline{true}) & = W \\
val_I(\underline{false}) & = F \\
val_I(p) & = I(p) \text{ für } p \in \Sigma \\
val_I(\neg p) & = \begin{cases} W & \text{falls } I(p) = F \\ F & \text{falls } I(p) = W \end{cases} \\
val_I(\alpha) & = \begin{cases} W & \text{falls } val_I(\alpha_1) = W \text{ und } val_I(\alpha_2) = W \\ F & \text{falls } val_I(\alpha_1) = F \text{ oder } val_I(\alpha_2) = F \end{cases} \\
val_I(\beta) & = \begin{cases} W & \text{falls } val_I(\beta_1) = W \text{ oder } val_I(\beta_2) = W \\ F & \text{falls } val_I(\beta_1) = F \text{ und } val_I(\beta_2) = F \end{cases} \\
val_I(A \leftrightarrow B) & = \begin{cases} W & \text{falls } val_I(A) = val_I(B) \\ F & \text{falls } val_I(A) \neq val_I(B) \end{cases}
\end{array}
$$

[1] Man beachte, daß eine Formel mehrfach in einer anderen als Unterformel auftreten kann. Damit die Polarität wohldefiniert ist, muß sie daher für das *Auftreten* einer Unterformel definiert werden.

■

Definition 2.8 *Eine Interpretation* I *über* Σ *heißt* Modell *für eine Formelmenge* $M \subset For0_\Sigma$*, in Symbolen* $I \models_\Sigma M$*, falls* $val_I(A) = W$ *für jedes* $A \in M$*.* A *bzw.* M *heißen dann* erfüllbar*.* A *ist* semantisch folgerbar *aus* M*, in Symbolen* $M \models_\Sigma A$*, falls jedes Modell von* M *auch Modell von* A *ist.* A *ist eine* Tautologie*, wenn jede Interpretation* I *auch Modell von* A *ist.* $A, B \in For0_\Sigma$ *heißen* logisch äquivalent*, falls* $\{A\} \models_\Sigma B$ *und* $\{B\} \models_\Sigma A$*.* ■

Als *Schreibweise* für Interpretationen verwenden wir oft die von I wahr gemachte Teilmenge von Σ, das heißt man identifiziert I mit der Aufzählung $\{p \in \Sigma \mid I(p) = W\}$.

Ist zum Beispiel $\Sigma = \{p_1, p_2, \dots\}$, $I(p_1) = I(p_3) = W$ und $I \equiv F$ sonst, so schreiben wir $I = \{p_1, p_3\}$.

Beispiel 2.9 Sei $M = \{p \vee q, \neg p\}$ und $I = \{q\}$. I ist ein Modell von M, $I' = \emptyset$ jedoch nicht, also ist M keine Tautologie. Die Notation $\{q\}$ für I ist nicht mit der *Formelmenge* $\{q\}$ zu verwechseln. Zum Beispiel gilt nicht $\{q\} \models_\Sigma M$. Will man I als Formelmenge hinschreiben, so ist $\{p \in \Sigma \mid I(p) = W\} \cup \{\neg p \in \Sigma \mid I(p) = F\}$ zu wählen, in dem Beispiel also $\{\neg p, q\}$, aus dem M natürlich wieder folgt. ■

Proposition 2.10 (Charakterisierung der Folgerbarkeit in der AL) *Seien* $A, B \in For0_\Sigma$ *und* $M \subset For0_\Sigma$*.*

Dann gilt $M \models_\Sigma A$ *gdw* $M \cup \{\neg A\}$ *nicht erfüllbar ist; insbesondere gilt:* A *ist eine AL-Tautologie gdw* $\models_\Sigma A$ *gdw* $\neg A$ *nicht erfüllbar ist. Weiterhin gilt, daß* A *und* B *logisch äquivalent sind gdw* $\models_\Sigma A \leftrightarrow B$*; und es gilt* $\{A\} \models_\Sigma B$ *gdw* $\models A \to B$ *(Deduktionstheorem).*

Die Signifikanz dieser Proposition ist, daß sie eine Problemreduktion erlaubt: um die Folgerbarkeitsrelation zu charakterisieren reicht es aus, eine Formel auf ihre Erfüllbarkeit testen zu können. Das wird im Automatischen Beweisen oft ausgenutzt.

Proposition 2.11 (AL-Ersetzungstheorem) *Gegeben seien* A, B, C, $D \in For0_\Sigma$*, so daß* $\models C \leftrightarrow D$*. Ferner sei* B *aus* A *entstanden, indem beliebige Vorkommen von* C *in* A *durch* D *ersetzt werden. Dann sind* A, B *logisch äquivalent.*

2.2 Prädikatenlogik (PL)

Definition 2.12 *Die Menge der* Sonderzeichen *der AL wird um die folgenden erweitert: „,“, „*$\forall$*“, „*$\exists$*“ sowie eine abzählbare Menge von* **Objektvariablen** $Var = \{x_0, x_1, \dots\}$*.*

Eine prädikatenlogische (PL) Signatur Σ *ist ein Tripel* $\langle F_\Sigma, P_\Sigma, \alpha_\Sigma \rangle$, *wobei* F_Σ, P_Σ *disjunkte, abzählbare Mengen von* ***Funktions-***, *bzw.* ***Prädikatsymbolen*** *sind und* $\alpha_\Sigma : F_\Sigma \cup P_\Sigma \to \mathbb{N}$ *deren Stelligkeit angibt. Wir verlangen, daß* $P_\Sigma \neq \emptyset$. ■

Die Bedingung $P_\Sigma \neq \emptyset$ garantiert nur, daß es nichttriviale Formeln über Σ gibt.

Beispiel 2.13

$$\Sigma_\leq = \langle \{0, a, b, f\}, \{\text{in\_iv}, \text{leq}\}, \alpha \rangle \quad ,$$

wobei $\alpha(0) = \alpha(a) = \alpha(b) = 0$, $\alpha(f) = 1$, $\alpha(\text{leq}) = 2$ und $\alpha(\text{in\_iv}) = 3$. Die Funktion α notieren wir manchmal auch als Menge von Urbild/Funktionswert-Paaren:

$$\alpha = \{\langle 0, 0 \rangle, \langle a, 0 \rangle, \langle b, 0 \rangle, \langle f, 1 \rangle, \langle \text{leq}, 2 \rangle, \langle \text{in\_iv}, 3 \rangle\}$$

■

Definition 2.14 *Die Menge der Terme* $Term_\Sigma$ *ist induktiv über* Σ *definiert als die kleinste Menge mit:*

1. $Var \subseteq Term_\Sigma$.
2. *Wenn* $f \in F_\Sigma$, $\alpha_\Sigma(f) = n$, $t_1, \ldots, t_n \in Term_\Sigma$, *dann auch* $f(t_1, \ldots, t_n) \in Term_\Sigma$.

Mit $Var(t)$ *bezeichnen wir die in* $t \in Term_\Sigma$ *vorkommenden Variablen. Ist* $Var(t) = \emptyset$, *so heißt* t Grundterm. *Mit* $Term_\Sigma^0$ *bezeichnen wir die Menge der Grundterme. Ein* Unterterm *von* t *ist ein Teilwort von* t, *das selbst Term ist.* ■

Beispiel 2.15 $f(x), f(f(0)) \in \text{Term}_{\Sigma_\leq}$, $a, f(f(0)) \in \text{Term}^0_{\Sigma_\leq}$. $\text{Var}(f(x)) = \{x\}$, die Unterterme von $f(f(0))$ sind: $\{0, f(0), f(f(0))\}$. ■

Definition 2.16 *Die Menge* At_Σ *der (PL)* ***Atome*** *ist definiert als*

$$At_\Sigma = \{p(t_1, \ldots, t_n) | p \in P_\Sigma,\ \alpha_\Sigma(p) = n, t_1, \ldots, t_n \in Term_\Sigma\}$$

Damit wird die Menge For_Σ *der* ***PL-Formeln*** *über* Σ *definiert als die kleinste Menge mit folgenden Eigenschaften:*

1. $\underline{true}, \underline{false} \in For_\Sigma$, $At_\Sigma \subset For_\Sigma$.
2. *Wenn* $A, B \in For_\Sigma$, $x \in Var$ *dann sind auch* $\neg A$, $A \wedge B$, $A \vee B$, $A \to B$, $A \leftrightarrow B$, $\forall x A$, $\exists x A \in For_\Sigma$.

Formeln der Gestalt $\neg p$, p *mit* $p \in At_\Sigma$ *heißen (PL)* Literal. ■

In Formeln der Gestalt $\forall xA, \exists xA$ heißt A der *Wirkungsbereich* oder Skopus des Quantors $\forall$ bzw. $\exists$. Das Auftreten von x in A heißt *gebunden*, falls es innerhalb des Wirkungsbereichs eines Quantors für x stattfindet, es heißt *frei*, falls dem nicht so ist und unmittelbar links von x kein Quantorsymbol steht. Dafür werden Abkürzungen eingeführt:

$$\begin{aligned} \mathrm{Bd}(A) &= \{x \in \mathrm{Var} \mid x \text{ tritt gebunden in } A \text{ auf}\} \\ \mathrm{Frei}(A) &= \{x \in \mathrm{Var} \mid x \text{ tritt frei in } A \text{ auf}\} \\ \mathrm{Var}(A) &= \mathrm{Bd}(A) \cup \mathrm{Frei}(A) \end{aligned}$$

Eine Formel $A \in \mathrm{For}_\Sigma$ heißt *abgeschlossen*, falls $\mathrm{Frei}(A) = \emptyset$.

Ein kleines technisches Problem mit diesen Definitionen ist, daß es Variablen geben kann, die weder gebunden noch frei in A auftauchen, aber trotzdem in A vorkommen, nämlich wie x in Unterformeln der Art QxB, wobei $Q \in \{\exists, \forall\}$ und x nicht im Wirkungsbereich von Qx vorkommt. Wie wir gleich sehen werden kann man aber in diesem Fall QxB durch B ersetzen, ohne die Bedeutung von A zu verändern, weswegen wir solche Formeln ohne Beschränkung der Allgemeinheit ausschließen.

Beispiel 2.17 Sei

$$\phi = \neg \mathrm{leq}(y, x) \rightarrow \exists z(\neg \mathrm{leq}(z, x) \wedge \neg \mathrm{leq}(y, z)) \ . \tag{2.1}$$

Dann ist die Unterformel $\mathrm{leq}(y, x)$ in $At_{\Sigma_\leq}$, $\neg\mathrm{leq}(y, x)$ ist ein Literal und $\neg\mathrm{leq}(z, x) \vee \neg\mathrm{leq}(y, z)$ ist im Wirkungsbereich von $\exists z$. $\psi = \forall x, y\phi$ ist abgeschlossen. Weiterhin ist $\mathrm{Bd}(\phi) = \{z\}$, $\mathrm{Frei}(\phi) = \{x, y\}$, sowie $\mathrm{Var}(\phi) = \{x, y, z\}$. ■

Definition 2.18 *Die* ***uniforme Notation*** *wird wie in Tabelle 2.1 unten dargestellt auf PL-Formeln erweitert. Dabei sind Formeln der Gestalt* $\forall xA$, $\neg\exists xA$ *vom* ***Typ*** γ *bzw.* ***universell****. Formeln der Gestalt* $\neg\forall xA$, $\exists xA$ *sind vom* ***Typ*** δ *bzw.* ***existentiell****.* ■

δ_1 und γ_1 ist noch die quantifizierte Variable als Argument beigegeben, weil man diese oft explizit braucht.

Definition 2.19 *Eine* Substitution *ist eine Abbildung* $\sigma : Var \rightarrow Term_\Sigma$. *Wir verwenden die Schreibweise* $\phi\sigma$ *anstatt* $\sigma(\phi)$ *für die Anwendung von Substitutionen, sowie* $\sigma = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ *falls* $\sigma(x_i) = t_i$ *für* $1 \leq i \leq n$ *und* $\sigma(y) = y$ *sonst.*

σ *wird auf* $Term_\Sigma \rightarrow Term_\Sigma$ *und* $For_\Sigma \rightarrow For_\Sigma$ *bzw.* $2^{Term_\Sigma} \rightarrow 2^{Term_\Sigma}$ *und* $2^{For_\Sigma} \rightarrow 2^{For_\Sigma}$ *fortgesetzt vermöge:*

$$\begin{aligned}
\sigma(c) &= c \\
\sigma(s(t_1,\ldots,t_n)) &= s(\sigma(t_1),\ldots,\sigma(t_n)),\ \ s \in F_\Sigma \cup P_\Sigma \\
\sigma(A \bullet B) &= \sigma(A) \bullet \sigma(B),\ \ \bullet \in \{\neg,\wedge,\vee,\rightarrow,\leftrightarrow\} \\
\sigma(QxA) &= Qx\sigma'(A),\ \ \sigma' = \sigma\backslash\{x \leftarrow t | t \in \mathit{Term}_\Sigma\} \\
\sigma(\{A_1,\ldots,A_n\}) &= \{\sigma(A_1),\ldots,\sigma(A_n)\}
\end{aligned}$$

Wir definieren $(\sigma \circ \tau)(x) = \sigma(\tau(x))$. *Eine Substitution, die eine Permutation auf Var ist, heißt* Variablenumbenennung. ■

In Zukunft steht der Ausdruck $\delta_1(t)$ für $(\delta_1(x))\{x \leftarrow t\}$ und ähnlich bei γ.

Die Fortsetzung von Substitutionen auf Term_Σ enthält übrigens ein technisches Problem: falls $z \in \mathrm{Bd}(\phi)$, $y \in \mathrm{Frei}(\phi)$ und $\sigma(y) = z$ wird y in ϕ durch Anwendung von σ gebunden, zum Beispiel passiert dies mit ϕ aus (2.1). Dieser Effekt ist unerwünscht kann jedoch leicht verhindert werden, wofür folgende technische Definition gebraucht wird:

Definition 2.20 *Sei* $x \in \mathit{Bd}(A), y \notin \mathit{Var}(A), A \in \mathit{For}_\Sigma$. *B entstehe aus A, indem alle Vorkommen von x in A durch y ersetzt werden. Dann sagen wir, daß B durch* gebundene Umbenennung *aus A entsteht.* ■

In Zukunft gehen wir also davon aus, daß der kritische Fall durch gebundene Umbenennung der gebundenen Variablen in geeigneter Weise vor Anwendung einer Substitution verhindert wird.

Beispiel 2.21 Sei $\sigma = \{x \leftarrow a, y \leftarrow w, z \leftarrow c\}$ und ϕ aus (2.1). Dann ist

$$\phi\sigma = \neg\mathrm{leq}(w,a) \rightarrow \exists z(\neg\mathrm{leq}(z,a) \wedge \neg\mathrm{leq}(w,z))\ .$$

Man beache, daß nur freie Vorkommen von Variablen ersetzt werden. ■

Definition 2.22 *Zwei Formeln* A, $B \in \mathit{For}_\Sigma$ *sind ein* komplementäres Paar, *falls* $\neg A$ *bis auf (mehrfache) gebundene Umbenennung identisch mit B ist.* ■

Beispiel 2.23 $\forall x\mathrm{leq}(x,a)$ und $\neg\forall y\mathrm{leq}(y,a)$ bilden ein komplementäres Paar. ■

Definition 2.24 *Sei* $\emptyset \neq T \subset \mathit{Term}_\Sigma$ *und* σ *eine Substitution.* σ *ist ein* Unifikator *von T gdw die Menge* $\sigma(T)$ *genau ein Element enthält.* μ *ist ein* allgemeinster Unifikator (mgu)[2] *von T gdw* μ *ein Unifikator von T ist und für alle Unifikatoren* σ *von T eine Substitution* ρ *existiert, so daß* $\sigma = \rho \circ \mu$. ■

[2] Engl.: most general unifier.

Beispiel 2.25 $\{x \leftarrow c, y \leftarrow a, z \leftarrow c\}$ ist ein Unifikator von $\{f(x,a), f(z,y)\}$, aber kein mgu. Ein mgu wäre $\{x \leftarrow z, y \leftarrow a\}$. $\{f(x,x), f(y,f(y))\}$ ist nicht unifizierbar, da $y \neq f(y)$ für alle y. ∎

Proposition 2.26 *Ist $\emptyset \neq T \subset Term_\Sigma$ unifizierbar, so gibt es auch einen idempotenten*[3] *mgu von T, der bis auf Variablenumbenennung eindeutig*[4] *festgelegt ist. Weiterhin gibt es einen Algorithmus, der in $\mathcal{O}(|T|)$ Schritten entweder den mgu von T in einer geeigneten Repräsentation ausgibt oder die Antwort „nicht unifizierbar“ ausgibt.*

Ein Unifikationsalgorithmus wie er von obiger Proposition gefordert wird, ist zum Beispiel in Martelli and Montanari [1982] beschrieben. Die lineare Komplexitätsaussage gilt nur, falls die unifizierende Substitution in geeigneter Weise repräsentiert wird.

Definition 2.27 *Relativ zu einer Signatur Σ wird eine* prädikatenlogische Interpretation[5] $\mathcal{D} = \langle D, I \rangle$ *angegeben, wobei $D \neq \emptyset$ eine beliebige Menge ist,* Universum *oder* Domain *genannt; für $f \in F_\Sigma$ ist $I(f)$ eine Abbildung* $I(f) : D^{\alpha(f)} \to D$, *für $p \in P_\Sigma$ ist $I(p)$ eine Abbildung $I(p) : D^{\alpha(p)} \to \{W, F\}$.*

Eine Variablenbelegung *über $\mathcal{D}$ ist eine Abbildung $\beta : Var \to D$. Wir definieren die* Modifikation von β *an der Stelle x zu d als:*

$$\beta_x^d(y) = \begin{cases} d & \textit{falls } x = y \\ \beta(y) & \textit{sonst} \end{cases}$$

Jede PL-Interpretation $\mathcal{D}$ und Variablenbelegung β induzieren eine Auswertungsfunktion *auf Termen und PL-Formeln $val_{\mathcal{D},\beta}$, wobei $val_{\mathcal{D},\beta} : For_\Sigma \to \{W, F\}$ und $val_{\mathcal{D},\beta} : Term_\Sigma \to D$, vermöge der Vorschrift:*

$$val_{\mathcal{D},\beta}(x) = \beta(x) \textit{ für } x \in Var$$

$$val_{\mathcal{D},\beta}(f(t_1, \ldots, t_n)) = I(f)(val_{\mathcal{D},\beta}(t_1), \ldots, val_{\mathcal{D},\beta}(t_n))$$
für $f(t_1, \ldots, t_n) \in Term_\Sigma$

$$val_{\mathcal{D},\beta}(\underline{true}) = W$$

$$val_{\mathcal{D},\beta}(\underline{false}) = F$$

$$val_{\mathcal{D},\beta}(p(t_1, \ldots, t_n)) = I(p)(val_{\mathcal{D},\beta}(t_1), \ldots, val_{\mathcal{D},\beta}(t_n))$$
für $p(t_1, \ldots, t_n) \in At_\Sigma$

$$val_{\mathcal{D},\beta}(\neg A) = \begin{cases} W & \textit{falls } val_{\mathcal{D},\beta}(A) = F \\ F & \textit{falls } val_{\mathcal{D},\beta}(A) = W \end{cases}$$

[3] Eine Abbildung σ ist **idempotent**, falls $\sigma \circ \sigma = \sigma$.
[4] Das heißt zu allen mgus σ, σ' gibt es eine Variablenumbenennung ν, so daß $\sigma\nu = \sigma'$.
[5] PL-Interpretationen werden in manchen Büchern *Strukturen* genannt.

$$val_{\mathcal{D},\beta}(\alpha) = \begin{cases} W & \textit{falls } val_{\mathcal{D},\beta}(\alpha_1) = W \textit{ und } val_{\mathcal{D},\beta}(\alpha_2) = W \\ F & \textit{falls } val_{\mathcal{D},\beta}(\alpha_1) = F \textit{ oder } val_{\mathcal{D},\beta}(\alpha_2) = F \end{cases}$$

$$val_{\mathcal{D},\beta}(\beta) = \begin{cases} W & \textit{falls } val_{\mathcal{D},\beta}(\beta_1) = W \textit{ oder } val_{\mathcal{D},\beta}(\beta_2) = W \\ F & \textit{falls } val_{\mathcal{D},\beta}(\beta_1) = F \textit{ und } val_{\mathcal{D},\beta}(\beta_2) = F \end{cases}$$

$$val_{\mathcal{D},\beta}(A \leftrightarrow B) = \begin{cases} W & \textit{falls } val_{\mathcal{D},\beta}(A) = val_{\mathcal{D},\beta}(B) \\ F & \textit{falls } val_{\mathcal{D},\beta}(A) \neq val_{\mathcal{D},\beta}(B) \end{cases}$$

$$val_{\mathcal{D},\beta}(\forall x A) = \begin{cases} W & \textit{falls } val_{\mathcal{D},\beta^d_x}(A) = W \textit{ für alle } d \in D \\ F & \textit{sonst} \end{cases}$$

$$val_{\mathcal{D},\beta}(\exists x A) = \begin{cases} W & \textit{falls } val_{\mathcal{D},\beta^d_x}(A) = W \textit{ für ein } d \in D \\ F & \textit{sonst} \end{cases}$$

■

Man beachte, daß für 0-stellige Funktionen (Konstanten) c bzw. 0-stellige Prädikate p gilt: $I(c) \in (D^0 \to D) \simeq D$ bzw. $I(p) \in (D^0 \to \{W, F\}) \simeq \{W, F\}$. Im letzteren Falle ist I also auch eine AL-Interpretation auf solchen Prädikaten.

Beispiel 2.28 Sei Σ die Signatur $\Sigma_{\leq}$ aus Beispiel 2.13. Wir definieren $\mathcal{D}_{\leq}$ über Σ wie folgt:

$$\begin{array}{lcl} D & = & \mathbb{R} \\ I(0) & = & 0 \\ I(a) & = & -1 \\ I(b) & = & 1 \\ I(f) & = & \begin{cases} \mathbb{R} \to \mathbb{R} \\ x \mapsto x^2 \end{cases} \\ I(leq(x,y)) = W & \text{gdw} & x \leq_{\mathbb{R}} y \\ I(in\_iv(a,x,b)) = W & \text{gdw} & x \in [a,b] \end{array}$$

Unter der Interpretation $\mathcal{D}_{\leq}$ bedeutet die Formel ψ aus Beispiel 2.17 die Dichte der reellen Zahlen. ■

Wie man im Beispiel sieht, sind die Konstanten von Σ und das Universum D im allgemeinen unterschiedliche Mengen.

Der folgende Satz zeigt, daß beliebige Substitutionen durch geeignete Belegungen ausgedrückt werden können.

Proposition 2.29 (Substitutionstheorem) *Für alle Signaturen Σ, Interpretationen $\mathcal{D}$, Belegungen β, Substitutionen σ, Terme $t \in Term_\Sigma$ und Formeln $A \in For_\Sigma$ gilt:*

1. $val_{\mathcal{D},\beta}(\sigma(t)) = val_{\mathcal{D},\beta'}(t)$

2. $val_{\mathcal{D},\beta}(\sigma(A)) = val_{\mathcal{D},\beta'}(A)$

falls $\beta'(x) = val_{\mathcal{D},\beta}(\sigma(x))$ *für alle* $x \in Var$.

Definition 2.30 *Eine PL-Interpretation* $\mathcal{D}$ *über* Σ *heißt* Modell *für eine Formelmenge* $M \subset For_\Sigma$, *in Symbolen* $\mathcal{D} \models_\Sigma M$, *falls* $val_{\mathcal{D},\beta}(A) = W$ *für jede Belegung* β *und jedes* $A \in M$. M *heißt* erfüllbar, *falls es eine Belegung* β *und eine Interpretation* $\mathcal{D}$ *gibt, so daß* $val_{\mathcal{D},\beta}(M) \subseteq W$, *kurz als* $\mathcal{D}, \beta \models M$ *geschrieben.*

A *ist* semantisch folgerbar *aus* M, *in Symbolen* $M \models_\Sigma A$, *falls jedes PL-Modell von* M *auch PL-Modell von* A *ist.*

A *ist* allgemeingültig, *in Symbolen* $\models_\Sigma A$, *wenn jede Interpretation* $\mathcal{D}$ *ein Modell von* A *ist.*

$A, B \in For_\Sigma$ *heißen* logisch äquivalent *falls* $\models_\Sigma A \leftrightarrow B$. *Sie heißen* erfüllbarkeitsäquivalent *falls* A *erfüllbar ist gdw* B *erfüllbar ist.*

$A \in For_\Sigma$ *ist* ***Tautologie*** *falls es eine* AL-*Signatur* $\Sigma' \subseteq For_\Sigma$ *gibt, so daß* A *AL-Tautologie über* Σ' *ist.* ■

Beispiel 2.31 $\mathcal{D}_{\leq}$ ist ein Modell der Formel ψ aus Beispiel 2.17, da $\mathbb{R}$ eine dichte Menge ist.

Die Formel $\forall x p(x) \rightarrow \exists y p(y)$ ist allgemeingültig aber keine Tautologie. Die Formel $\forall x p(x) \vee \neg\forall x p(x)$ ist eine Tautologie, da sie eine AL-Tautologie über $\Sigma' = \{\forall x p(x)\}$ ist. ■

Proposition 2.32 *Seien* $A, B \in For_\Sigma$, $M \subset For_\Sigma$. *Falls* M *abgeschlossen ist, dann gilt* $M \models_\Sigma A$ *gdw* $M \cup \{\neg A\}$ *ist nicht erfüllbar; insbesondere gilt:* A *ist allgemeingültig gdw* $\neg A$ *nicht erfüllbar ist. Jede Tautologie ist auch allgemeingültig.*

Wie das Beispiel $M = \{p(x)\}$, $A = \forall x p(x)$ zeigt, ist die Abgeschlossenheit von M in obiger Proposition wesentlich. Es gilt zwar $M \models A$, aber $\{p(x), \neg\forall x p(x)\}$ ist erfüllbar, zum Beispiel mit $\mathcal{D} = \langle\{a, b\}, I\rangle$, $I(p) = \{b\}$, $\beta(x) = b$.

Proposition 2.33 *Gegeben seien* $A, B, C, D \in For_\Sigma$, *so daß* $\models C \leftrightarrow D$. *Ferner sei* B *aus* A *entstanden, indem beliebige Vorkommen von* C *in* A *durch* D *ersetzt werden. Dann sind* A, B *logisch äquivalent.*

2.3 Formalisierung in Logik erster Stufe: Ein Beispiel aus der Analysis

Das folgende Beispiel ist im wesentlichen Wang and Bledsoe [1987] entnommen.

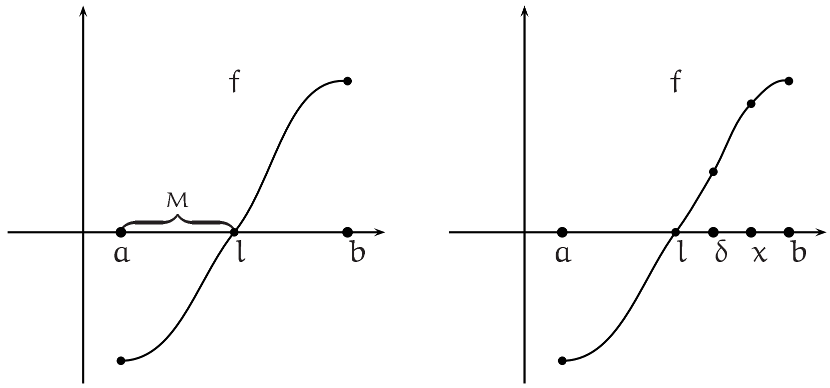


Abb. 2.1: Illustration des Nullstellensatzes.

Nullstellen- bzw. Zwischenwertsatz: Jede auf einem abgeschlossenen Intervall $[a, b]$ stetige Funktion f nimmt den Wert 0 an, falls $f(a) \leq 0$ und $f(b) \geq 0$.

Beweisidee: Betrachte das Supremum l der Menge $M = \{x \mid f(x) \leq 0, a \leq x \leq b\}$ und zeige, daß $f(l) = 0$, siehe Abbildung 2.1 links.

Formalisierungsidee: Axiomatisierung der totalen Ordnung auf den rellen Zahlen. *Das einzig benötigte Prädikat ist '$\leq$' !*

Abkürzungen: $x \in [a, b]$ gdw $a \leq x$ und $x \leq b$. $x < y$ gdw $y \not\leq x$. $x > y$ gdw $x \not\leq y$. $x \geq y$ gdw $y \leq x$.

Dichte: Wenn $x < y$, dann gibt es ein z mit $x < z$ und $z < y$.

Rechtsseitige Stetigkeit, positive Funktionste: Sei $x \in [a, b]$ sowie $f(x) > 0$. Dann gibt es ein $\delta < x$, so daß für alle $\delta < y \leq x$ immer noch $f(y) > 0$ gilt,[6] siehe Abbildung 2.1 rechts.

Linksseitige Stetigkeit, negative Funktionswerte: Sei $x \in [a, b]$ sowie $f(x) < 0$. Dann gibt es ein $\delta > x$, so daß für alle $\delta > y \geq x$ immer noch $f(y) < 0$ gilt.

l ***ist eine obere Schranke von*** M***:*** Wenn $l < x \leq b$, dann ist x nicht mehr in M, d.h. $f(x) > 0$. (Die Kontraposition davon ist: falls $x \in M$, dann $x \leq l$, die übliche Bedingung.)

l ***ist kleinste obere Schranke:*** Für alle $x < l$ gibt es ein Element $\delta \in M$, das größer als x ist, d.h.: für alle $x < l$ gibt es ein $x < \delta \leq b$, so daß $f(\delta) \leq 0$.

[6]Man beachte, daß δ hier nicht die Größe der „δ-Umgebung“ um die Nullstelle ist, sondern deren rechter Rand.

Randbedingungen: $a \leq b$, $f(a) \leq 0$, $f(b) \geq 0$.

Behauptung: Es gibt ein ξ, so daß $0 \leq f(\xi) \leq 0$.

Die Formalisierung des Beispiels in Logik erster Stufe ist nun nicht mehr schwierig und ist in Abbildung 2.2 angegeben.

Die ausschließliche Verwendung von $\leq$ macht die Axiomatisierung kompakt. Auch die Art der verwendeten Definitionen sind wichtig. Eine Charakterisierung der Stetigkeit mit Folgen zum Beispiel wäre in Logik erster Stufe problematisch. Darüberhinaus ist die verwendete Definition der Stetigkeit sehr spezifisch dem Problem angepaßt. Alle Voraussetzungen werden im Beweis auch tatsächlich gebraucht und die Definition des Supremums von M beinhaltet eine wesentliche Beweisidee. Doch ohne die

- redundanzfreie Formulierung,
- die spezifische Formulierung der Stetigkeit,
- die Vorgabe der Beweisidee,

haben automatische Beweiser zur Zeit keine Chance.

2.4 Warum gerade Prädikatenlogik?

Thema dieser Vorlesung ist im wesentlichen das Automatische Beweisen in *Prädikatenlogik*.

Es gibt daneben eine Reihe anderer Logiken; grob können diese in ausdrucksschwächere Logiken (Aussagenlogik, temporale Aussagenlogik, modale Aussagenlogiken, u. a.) und ausdrucksstärkere Logiken (Prädikatenlogik höherer Ordnung, Dynamische Logik, u. a.) eingeteilt werden.

Die Forschung im Automatischen Beweisen konzentriert sich nun hauptsächlich auf die Prädikatenlogik, da sie sich als die ausdrucksstärkste Logik erweist, die man noch „vernünftig“ mit Automatischem Beweisen behandeln kann.

Dennoch hat das Automatische Beweisen in Aussagenlogik, temporaler Aussagenlogik und Modallogik einen wichtigen Stellenwert und bildet eigene Forschungsbereiche. Denn man sollte sich auf ausdrucksschwächere Logiken beschränken, wenn diese für eine konkrete Anwendung ausreichen. Viele Ergebnisse, die in der Vorlesung vorgestellt weden, lassen sich auf das Automatische Beweisen in schwächeren Logiken übertragen. Gerade die Entscheidbarkeit dieser Logiken bedingt aber auch Unterschiede.

Stärkere Logiken kann man nur beschränkt automatisch behandeln. Dann werden Dinge wie Ergonomie der interaktiven Beweis-Systeme wichtig. Aber auch in diesem Bereich ist die Erhöhung des Automatisierungsgrades von Interesse.

```
% Intervalle und Randbedingungen
∀l,x,u((leq(l,x)∧leq(x,u))→in_iv(l,x,u))
leq(a,b)
leq(f(a),0)
leq(0,f(b))

% Totale Ordnung
∀x leq(x,x)
∀x,y(leq(x,y)∨leq(y,x))
∀x,y,z((leq(x,y)∧leq(y,z))→leq(x,z))

% Dichtheit
∀x,y(¬leq(y,x)→ ∃z(¬leq(z,x)∧¬leq(y,z)))

% Stetigkeit von f
∀x((in_iv(a,x,b)∧¬leq(f(x),0))→
    ∃δ(¬leq(x,δ)∧∀y((¬leq(y,δ)∧leq(y,x))→¬leq(f(y),0))))

∀x((in_iv(a,x,b)∧¬leq(0,f(x)))→
    ∃δ(¬leq(δ,x)∧∀y((¬leq(δ,y)∧leq(x,y))→¬leq(0,f(y)))))

% l ist Supremum von M
∀x((¬leq(x,l)∧leq(x,b))→¬leq(f(x),0))
∀x(¬leq(l,x)→∃δ(leq(δ,b)∧¬leq(δ,x)∧leq(f(δ),0)))

% Behauptung
∃ξ in_iv(0,f(ξ),0)
```

Abb. 2.2: Formalisierung des Zwischenwertsatzes in Logik erster Stufe.

2.5 Metaresultate

Mit Metaresultaten werden Theoreme bezeichnet, die etwas *über* eine Logik sagen, im Gegensatz zu solchen Aussagen, die man *innerhalb* einer Logik beweisen kann. Das vielleicht berühmteste Metaresultat ist der erste Gödelsche Unvollständigkeitssatz, der die Unmöglichkeit einer vollständigen Axiomatisierung der Arithmetik in PL erster Stufe zum Inhalt hat. In dieser Vorlesung befassen wir uns allerdings ausschließlich mit vollständigen Axiomatisierungen.

Satz 2.34 (Komplexität der AL und der PL) *Die Relation $\models$ ist in der AL entscheidbar. Insbesondere ist die Menge der AL-Tautologien co-NP-vollständig, die Menge der erfüllbaren AL-Formeln ist NP-vollständig.*

Die Relation $\models$ ist in der PL unentscheidbar. Insbesondere ist die Menge der allgemeingültigen PL-Formeln rekursiv aufzählbar, die Menge aller erfüllbaren PL-Formeln jedoch nicht.

Das für das Automatische Beweisen wohl wichtigste Metaresultat ist der Satz von Herbrand.

Definition 2.35 *Eine PL-Interpretation $\mathcal{D} = \langle D, I \rangle$ ist eine* Term-Interpretation *falls* $D = \mathit{Term}^0_\Sigma$. ■

Definition 2.36 *Es sei $\mathit{Term}^0_\Sigma \neq \emptyset$.*

$\mathcal{D} = \langle D, I \rangle$ *ist eine* Herbrand-Interpretation, *falls*

1. $D = \mathit{Term}^0_\Sigma$ *($\mathcal{D}$ also eine Term-Interpretation ist), und*
2. $I(f)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ *für alle* $f \in F_\Sigma$, $t_1, \ldots, t_n \in \mathit{Term}^0_\Sigma$.

Eine Herbrand-Interpretation, die ein Modell für $M \subseteq \mathit{For}_\Sigma$ ist, heißt Herbrand-Modell *für* M. ■

In einer Herbrand-Interpretation ist jeder Grund-Term seine eigene Interpretation.

Definition 2.37 *Sei M eine Menge geschlossener PNF-Formeln ohne Existenzquantoren (siehe Def. 2.44). Die Menge der* T-Instanzen *von M ist definiert als:*

$$T(M) = \{B\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\} \mid A \in M,\ B \text{ Matrix von } A,\\ \mathit{Frei}(B) = \{x_1, \ldots, x_n\},\ t_1, \ldots, t_n \in T\}$$

■

Satz 2.38 (Satz von Herbrand) *Sei $\mathit{Term}^0_\Sigma \neq \emptyset$ und M eine Menge geschlossener PNF-Formeln ohne Existenzquantoren.*

Dann hat M ein Modell gdw M ein Herbrand-Modell hat gdw $\mathit{Term}^0_\Sigma(M)$ ein Modell hat.

Beweisskizze: Konstruiere aus einem beliebigen (Term-)Modell ein Herbrand-Modell vermittels

$$I_H(p)(t) = I(p)(I(t))$$

■

Ein weiteresss äußerst wichtiges Meta-Resultat ist der Kompaktheitssatz:

Satz 2.39 (Kompaktheit der PL) *$M \subseteq For_\Sigma$ hat ein Modell gdw alle endlichen Teilmengen von M ein Modell haben.*

Der Kompaktheitssatz folgt aus der später bewiesenen Vollständigkeit der Kalküle, deren Beweis den Kompaktheitssatz nicht verwendet. Ein direkter Beweis findet sich im Skriptum zur Vorlesung „Formale Systeme".

Kombiniert man den Kompaktheitsatz 2.39 und den Satz von Herbrand, so erhält man folgende Variante des Satzes von Herbrand:

Satz 2.40 *Sei $Term^0_\Sigma \neq \emptyset$ und $M \subseteq For_\Sigma$ eine Menge geschlossener PNF-Formeln (Def. 2.44) ohne Existenzquantoren.*

M ist unerfüllbar gdw $T(M)$ unerfüllbar ist für eine endliche *Teilmenge T von $Term^0_\Sigma$.*

Beweis: M ist unerfüllbar gdw M kein Modell besitzt (da M geschlossen ist) gdw $Term^0_\Sigma(M)$ kein Modell hat (Satz von Herbrand) gdw es eine endliche Teilmenge M' von $Term^0_\Sigma(M)$ ohne Modell gibt (Kompaktheit).

Für ein geeignetes $T \subset Term^0_\Sigma$ gilt $M' \subseteq T(M)$, woraus folgt, daß $T(M)$ unerfüllbar ist. ■

Der obige Satz von Herbrand ist die theoretische Rechtfertigung für die Möglichkeit, automatische Beweiser zu bauen. Er besagt nämlich, daß – modulo Skolemisierung – um die Allgemeingültigkeit einer Formel nachzuweisen, keine anderen Modelle betrachtet werden müssen, als die durch die Signatur der betrachteten Formel gegebenen und auch davon nur eine endliche Teilmenge. Die Konsequenz ist, daß bei der Suche nach einem formalen Beweis niemals Funktionssymbole betrachtet werden müssen, die nicht entweder schon vorhanden sind oder durch Skolemisierung eingeführt wurden. Dies schränkt den Suchraum entscheidend ein und ermöglicht die Verwendung von Unifikation bei der Suche nach einem Beweis.

Alle im folgenden vorgestellten Methoden zur Beweisfindung können als konstruktiver Beweise von Satz 2.40 aufgefaßt werden, wobei es darum geht die Menge T möglichst effizient zu finden.

2.6 Normalformen

Die Berechnung von Normalformen ist im Automatischen Beweisen von zentraler Bedeutung. Zum einen fallen bei der Berechnung einer Normalform Informationen an, die später sowieso gebraucht werden. Zweitens ermöglicht eine Darstellung in Normalform die Erkennung und Beseitigung gewisser Redundanzen in einer Formel. Wie wir sehen werden, verlangen gewisse Inferenzregeln auch, daß die Eingabe in einer bestimmten Normalform vorliegt.

Es ist jedoch auch möglich, einen gesamten formalen Beweis als Herstellung einer Normalform anzusehen. Ohne zu diesem Zeitpunkt näher darauf einzugehen, halten wir fest, daß sich zum Beispiel ein vollständiges semantisches Tableau für eine Formel A als disjunktive Normalform von A interpretieren läßt und die Menge aller nicht subsumierten Resolventen einer Menge von Klauseln C kann man als Primimplikantendarstellung von C auffassen.

Es ist also im allgemeinen nicht sinnvoll, streng zwischen einem Vorverarbeitungsschritt und dem eigentlichen Beweis zu unterscheiden. Wie wir sehen werden, ist es gelegentlich sogar vorteilhaft, Inferenz- und Vereinfachungsschritte miteinander zu verzahnen.

Ein Unterschied, der implizit oft zwischen Inferenz- und Vereinfachungsregeln gemacht wird, ist, daß pure Vereinfachungsregeln i.a. eine geringere Berechnungskomplexität als zugehörige Inferenzregeln aufweisen. So ist beispielsweise Subsumtion von PL-Klauseln entscheidbar (genauer: NP-vollständig), die semantische Folgerbarkeit einer Klausel aus einer anderen aber nicht. In der AL ist die Vereinfachung einer Klauselmenge durch *forward chaining* mit linearem Zeitaufwand möglich, die Berechnung aller Primimplikanten ist jedoch NP-hart. Dies bedeutet natürlich, daß Vereinfachungsregeln auch unvollständig sind.

Definition 2.41 *$A \in For_\Sigma$ ist in* Negationsnormalform (NNF)*, falls Negationszeichen in A nur als Teil eines Literals vorkommen (also direkt vor einem Atom stehen) und der Implikations- und der Äquivalenzoperator ($\rightarrow$ und $\leftrightarrow$) in A nicht vorkommen.* ∎

Eine Formel A kann in eine äquivalente Formel A' in NNF transformiert werden, indem zuerst Unterformeln der Form $B \rightarrow B'$ durch $\neg B \vee B'$ und Unterformeln der Form $B \leftrightarrow B'$ durch $(B \wedge B') \vee (\neg B \wedge \neg B')$ ersetzt werden; dann werden Negationszeichen durch Anwendung der de Morganschen Regeln vor die Atome gezogen. Dabei eventuell auftretende Doppelnegationen werden eliminiert.

Enthält A keine Äquivalenzoperatoren, dann ist die Größe der durch die Transformation gebildeten Formel A' offenbar linear in der Größe von A; andernfalls kann sie exponentiall größer sein.

Definition 2.42 *$A \in For_\Sigma$ heißt* bereinigt*, falls gilt: $Frei(A) \cap Bd(A) = \emptyset$ und falls QxB, $Q'x'B'$ verschiedene Unterformeln von A sind, dann ist $x \neq x'$.* ∎

Eine zu einer Formel A äquivalente bereinigte Formel A' erhält man durch geeignete gebundene Umbenennung (mit linearem Aufwand).

Definition 2.43 *$A \in For_\Sigma$ ist in* konjunktiver Normalform (KNF), *falls A quantorenfrei ist und die Gestalt $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} L_{i,j}$ hat, wobei die $L_{i,j}$ Literale sind.* ∎

Eine zu einer Formel A äquivalente Formel A' in KNF kann man berechnen, indem man A zunächst in NNF transformiert und anschließend die Distributivitätsgesetze anwendet. Das Resultat A' kann exponentiell größer sein als A (auch wenn A keine Äquivalenzoperatoren enthält).

Definition 2.44 *$A \in For_\Sigma$ ist in* Pränexnormalform (PNF), *falls A die Gestalt $Q_1x_1 \cdots Q_nx_nB$ hat, wobei $Q \in \{\forall, \exists\}$, $x_1, \ldots, x_n \in Var$ und $B \in For_\Sigma$ eine quantorenfreie Formel ist. B heißt* Matrix *von A, $Q_1x_1 \cdots Q_nx_n$ heißt der* Präfix *von A.* ∎

Zur Berechnung einer zu einer Formel A äquivalenten Formel A' in PNF wird A zunächst in NNF transformiert und bereinigt. Dann werden alle Quantoren so lange nach „außen“ gezogen, bis sie auf einen anderen Quantor treffen, wobei man mit dem am weitesten außen stehenden beginnt.

Enthält A keine Äquivalenzoperatoren, dann – wie bei der Transformation in NNF – die Größe der durch resultierenden PNF-Formel linear in der Größe von A; andernfalls kann sie exponentiall größer sein.

Definition 2.45 *$A \in For_\Sigma$ ist in* Skolemnormalform (SNF)[7], *falls A in PNF ist, geschlossen, im Präfix von A nur universelle Quantoren auftauchen und die Matrix von A in KNF ist.* ∎

Zur Transformation in SNF wird eine Formel zunächst in PNF überführt, dann werden durch die übliche Skolemisierung alle Existenzquantoren eliminiert, schließlich wird die Matrix in KNF überführt.

Während alle übrigen Formeltransformationen eine zur Ausgansgsformel A logisch äquivalente Formeln A' produzieren, gilt dies für die Skolemisierung nicht; hier sind A und A' nur *erfüllbarkeisäquivalent*, d. h., A ist erfüllbar gdw A' erfüllbar ist. Beispielsweise sind $\exists x p(x)$ und $p(c)$ nicht äquivalent; jedoch ist $\exists x p(x)$ genau dann erfüllbar, wenn $p(c)$ erfüllbar ist.

Beispiel 2.46 Wir geben einige Beispiele aus Abbildung 2.2. Das Dichteaxiom hat nach Eliminierung der Implikation die Form

$$\forall x, y[\mathrm{leq}(y, x) \vee \exists z(\neg \mathrm{leq}(z, x) \wedge \neg \mathrm{leq}(y, z))]$$

[7]Vorsicht: Manche Autoren verwenden den Begriff „KNF“, um die Normalform zu bezeichnen, die wir SNF nennen.

und ist dann in NNF und auch bereinigt. Der quantorfreie Teil des zweiten Axioms der totalen Ordnung ist in KNF. Das ganze Axiom ist in PNF. Für Formeln in KNF führen wir folgende Schreibweise ein:

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} L_{i,j} = \begin{array}{ll} L_{1,1} \vee \cdots \vee L_{1,m_1} & \leftarrow \text{Klausel} \\ \cdots & \\ L_{n,1} \vee \cdots \vee L_{n,m_n} & \end{array}$$

Wie angedeutet, heißt eine Zeile einer KNF **(disjunktive)** *Klausel*.

Weiterhin werden Quantoren gleicher Art oft zusammengefaßt, d.h., man schreibt $\forall x, y$ anstatt $\forall x \forall y$. ■

Proposition 2.47 *Zu jedem* $A \in For_\Sigma$ *gibt es Formeln* B *in NNF,* C *bereinigt,* D *in PNF, falls* A *quantorenfrei,* E *in KNF und, falls* A *geschlossen,* F *in SNF, so daß* $A, \ldots, E$ *logisch äquivalent und* $A, \ldots, F$ *erfüllbarkeitsäquivalent. Weiterhin gilt* $|B|, |C| \in \mathcal{O}(|A|)$ *und* $|D|, |E|, |F| \in \mathcal{O}(2^{|A|})$.

Das oben skizzierte Verfahren zur Herstellung einer SNF hat eine Reihe von Nachteilen.

- Die Formel kann bei der Transformation in KNF exponentiall wachsen – enthält sie Äquivalenzen auch schon bei der Transformation in NNF.
- Die syntaktische Struktur der Formel geht bei Herstellung von KNF verloren.
- Bei Skolemisierung eingeführte Funktionssymbole haben mehr Argumente als notwendig, davor der Skolemisierung PNF hergestellt wird (siehe Beispiel 2.48).
- Es werden unnötig viele verschiedene Skolem-Symbole eingeführt (siehe Beispiel 2.49).

Beispiel 2.48 Betrachte die Formel $\forall x((\exists y p(y)) \vee q(x))$. Die Herstellung der PNF ergibt $\forall x \exists y (p(y) \vee q(x))$. Skolemisierung ergibt nun $\forall x (p(f(x)) \vee q(x))$. Es ist jedoch nicht nötig, die Skolemvariable x in die Unterformel $p(y)$ einzuführen, weil x vor der PNF gar nicht im Wirkungsbereich von $\exists y$ frei vorkam und damit eine erfüllende Belegung von $p(y)$ nicht von der Wahl von x abhängt. ■

Beispiel 2.49 Bei Skolemisierung der Formel $A = (\exists x p(x)) \vee (\exists y p(y))$ werden zwei verschiedene Skolem-Konstanten eingeführt; das Ergebnis ist $p(c) \vee p(d)$. Das ist unnötig, denn zur Skolemisierung gleicher existenzquantifizierter Unterformeln kann das gleich Skolem-Symbol verwendet werden. Also ist auch $p(c) \vee p(c)$ eine zu A erfüllbarkeitsäquivalente Formel. ■

Zur Umgehung der geschilderten Nachteile werden die folgenden Verbesserungen der Normalformenberechnung eingeführt:

1. Skolemisierung ohne vorher PNF herzustellen.

2. Feste Zuordnung von Skolem-Symbolen zu δ-Formeln. Das zugehörige Skolem-Symbol wird bei mehrfachem Auftreten einer δ-(Unter-)Formel wiederholt zu deren Skolemisierung verwendet.

3. Verwendung einer definitorischen (strukturerhaltenden) KNF-Transformation, deren Ergebnis polynomielle Größe hat (allerdings nur erfüllbarkeitsäquivalent und im allgemeinen nicht äquivalent ist).

2.6.1 Verbesserte Skolemisierung

Wir beschreiben zunächst eine verbesserten Skolemisierung, für die keine PNF hergestellt werden muß und die den δ-Formeln fest zugeordnete Skolem-Symbole verwendet.

Definition 2.50 *Zu $\phi \in For_\Sigma$ bezeichne $[\phi]$ die Äquivalenzklasse aller bis auf Umbenennung der freien Variablen in ϕ mit ϕ identischen Formeln.*
■

Definition 2.51 *Zu jeder Signatur Σ sei eine Erweiterung Σ^* definiert durch:*

$$\begin{array}{rcll} F_0 & = & F_\Sigma & \\ F_n & = & F_{n-1} \cup \{f_{[\phi]} \mid \phi \in F_{n-1}\} & \quad \textit{für } n \in \mathbb{N} \\ F_{\Sigma^*} & = & \bigcup\limits_{n \geq 0} F_n & \\ P_{\Sigma^*} & = & P_\Sigma & \end{array}$$

■

Definition 2.52 *Eine Term-Interpretation $\mathcal{D}$ über Σ^* heiße* kanonische Interpretation*, falls für alle Belegungen β und alle δ-Formeln δ über Σ^* gilt:*

$$\begin{array}{ll} \textit{Wenn } val_{\mathcal{D},\beta}(\delta) & = W \ , \\ \textit{dann } \ val_{\mathcal{D},\beta}(\delta_1(f_{[\delta]}(x_1, \ldots, x_n))) & = W \ , \end{array}$$

wobei $Frei(\delta) = \{x_1, \ldots, x_n\}$. ■

In kanonischen Interpretationen sind die Skolem-Symbole gerade „richtig“ interpretiert, so daß es möglich ist, mit diesen Symbolen zu Skolemisieren und dabei die Erfüllbarkeit in kanonischen Interpretationen zu erhalten.

Die Existenz kanonischer Interpretationen ist aber nicht selbstverständlich, sondern muß bewiesen werden.

Satz 2.53 *Sei $\phi \in For_\Sigma$ abgeschlossen und erfüllbar. Dann hat ϕ ein kanonisches Modell $\mathcal{D}^*$.*

Beweis: Sei $\mathcal{D}$ ein beliebiges Term-Modell von ϕ.

Zur Definition von $\mathcal{D}^*$ werde die Interpretation der $f_{[\phi]}$ wie folgt verändert (diese kommen nicht in ϕ vor); sonst bleibt das Modell unverändert.

Zunächst sei der Rang $rg : F_{\Sigma^*} \backslash F_\Sigma \to \mathbb{N}$ der Skolem-Symbole wie folgt definiert:

$$rg(f_{[\psi]}) = \begin{cases} 0 & \psi \text{ nicht Typ } \delta \\ n & \text{sonst, wobei } n \text{ die Schachtelungstiefe von } \delta\text{-} \\ & \text{Formeln in } \psi \text{ ist} \end{cases}$$

Die Modelle $(\mathcal{D}^n)_{n \geq 0}$ seien Einschränkungen von $\mathcal{D}^*$ auf Signaturen Σ^n, wobei Σ^n die Beschränkung von Σ^* auf Symbole in Σ und solche mit Rang $\leq n$ ist. Diese Modelle approximieren $\mathcal{D}^*$. Die Interpretation I^n in $\mathcal{D}^n$ der Symbole $f \in F_{\Sigma^n} \backslash F_\Sigma$ sei wie folgt festgelegt.

$n = 0$: $I^0(f)$ beliebig.

$n > 0$: Sei $f = f_{[\delta]}$ und $Frei(\delta) = \{x_1, \ldots, x_k\}$; dann sei für alle $b_1, \ldots, b_k \in D$ festgelegt:

Falls es β gibt mit $\beta(x_i) = b_i$ und $\mathcal{D}^n$, β erfüllt δ, wähle $c \in D$, so daß $\mathcal{D}^n$, β_x^c erfüllt $\delta_1(x)$ und setze $I^{n+1}(f)(b_1, \ldots, b_k) = c$. Sonst sei $I^{n+1}(f)(b_1, \ldots, b_k)$ beliebig.

Das von den $\mathcal{D}^n$ approximierte $\mathcal{D}^*$ ist offenbar kanonisch ist. ■

In der Formulierung von Satz 2.53 ist es wesentlich, daß ϕ eine Formel nur über Σ ist. Beispielsweise hat $\neg p(c_{[\exists x p(x)]})$ kein kanonisches Modell.

Die Formel aus Beispiel 2.48 läßt sich nun direkt skolemisieren mit dem Ergebnis $\forall x(p(c) \vee q(x))$, wobei $c = f_{[\exists y p(y)]}$.

Da die Existenz kanonischer Interpretationen nun garantiert ist, reicht folgender Satz für die Skolemisierung aus:

Satz 2.54 *A sei eine bereinigte Formel über Σ^*.*

A' entstehe aus A indem

(a) ein positives Vorkommen einer δ-Formel δ in A durch

$$\delta_1(f_{[\delta]}(y_1, \ldots, y_n))$$

ersetzt wird, wobei $Frei(\delta) = \{y_1, \ldots, y_n\}$, oder

(b) ein negatives Vorkommen einer γ-Formel γ in A durch

$$\gamma_1(f_{[\neg\gamma]}(y_1, \ldots, y_n))$$

ersetzt wird, wobei $Frei(\gamma) = \{y_1, \ldots, y_n\}$.

Dann gilt für alle Belegungen β *und alle kanonischen Interpretationen* $\mathcal{D}^*$*:*

$$\mathcal{D}^*, \beta \models A \textit{ gdw } \mathcal{D}^*, \beta \models A'.$$

Beweisskizze: Der Beweis gelingt durch einfache strukturelle Induktion über den Formelaufbau von A. Der einzige kritische Fall wird direkt von der Eigenschaft kanonischer Interpretationen abgedeckt. ∎

2.6.2 Definitorische KNF

Die Transformation in KNF kann zu einer exponentiell größeren Formel führen. Dies läßt sich vermeiden, wenn man sogenannte definitorische oder strukturerhaltende Transformationen verwendet. Die Idee ist dabei die folgende: sei A bereinigt; sei B eine Unterformel von A der Gestalt $B = Q_1 x_1 \cdots Q_n x_n C$, wobei direkt links von B kein Quantor steht und C kein Literal ist, so daß B möglichst weit unten im Syntaxbaum von A vorkommt. Nun wird die Signatur von A um ein neues m-stelliges Prädikat p_B erweitert, wobei $\mathrm{Frei}(B) = \{y_1, \ldots, y_m\}$. Schließlich werden alle Vorkommen von solchen B in A durch $p_B(y_1, \ldots, y_m)$ ersetzt und die Formel

$$\forall y_1 \cdots \forall y_m (p_B(y_1, \ldots, y_m) \circ B)$$

erzeugt. Dabei ist

$$\circ = \begin{cases} \rightarrow & \text{falls } \mathrm{pol}(B, A) = + \\ \leftarrow & \text{falls } \mathrm{pol}(B, A) = - \\ \leftrightarrow & \text{falls } \mathrm{pol}(B, A) = \pm \end{cases}$$

Dieser Prozeß wird wiederholt, bis die einzige Unterformel der geforderten Gestalt A selbst ist. Zum Schluß wird die Formel A selbst (mit den ausgeführten Ersetzungen) erzeugt.

Jede der generierten Formeln enthält höchstens zwei binäre Operatoren. Man kann also jede dieser Formeln wie oben beschrieben mit konstantem Aufwand in eine logisch äquivalente SNF verwandeln. Insgesamt wurden höchstens $|A|$ Formeln erzeugt, die einen Präfix der Länge $\mathcal{O}(|A|)$ haben (Länge der Skolemfunktionen beachten!) und eine Matrix, die durch eine Konstante begrenzt ist. Die Konjunktion aller dieser Formeln hat also Länge $\mathcal{O}(|A|^2)$. Nach Bereinigung können alle Quantoren nach oben geschoben werden und das Endresultat ist in SNF.

Das Verfahren wurde für die AL ursprünglich von Tseitin (1970) angegeben. Die PL-Version mit einigen Optimierungen findet sich zum Beispiel in Plaisted and Greenbaum [1986].

Beispiel 2.55 Gegeben sei

$$A = \forall w \exists x (p(w,x) \vee \overbrace{\forall y (q(w,x,y) \wedge \exists z r(y,z))}^{B})$$

Die erste und einzige Ersetzung ist

$$\forall w \forall x (\boxed{s(w,x)} \rightarrow \boxed{\forall y (q(w,x,y) \wedge \exists z r(y,z))})$$

Man beachte, daß die Polarität der ersetzten Unterformel B positiv ist. Nach Skolemisierung der Umbenennung ergibt sich:

$$\forall w \forall x (s(w,x) \rightarrow \forall y (q(w,x,y) \wedge r(y,g(y))))$$

Verschieben des Allquantors und die Transformation in KNF ergibt:

$$\forall w \forall x \forall y ((\neg s(w,x) \vee q(w,x,y)) \wedge (\neg s(w,x) \vee r(y,g(y))))$$

Die ursprüngliche Formel A wird zu

$$\forall w \exists x (p(w,x) \vee \boxed{s(w,x)})$$

Nach Skolemisierung ergibt das:

$$\forall w (p(w,h(w)) \vee s(w,h(w)))$$

■

Zum Schluß dieses Abschnitts wollen wir noch eine gängige Schreibweise für Formeln in SNF angeben. Die besteht darin, die Kommutativität und Assoziativität der Konjunktion und Disjunktion auszunutzen und die Matrix einer Formel in SNF als eine endliche Familie endlicher Mengen von Literalen zu schreiben. Da im Präfix einer SNF-Formel nur universelle Quantoren vorkommen, können wir mit der Konvention, daß alle freien Variablen implizit allquantifiziert sind, auf das Hinschreiben der Quantoren verzichten. Die SNF, die sich aus dem Beispiel von oben ergibt, sieht dann (nach Bereinigung) folgendermaßen aus:

$$\begin{array}{l} \{p(w_1,h(w_1)), s(w_1,h(w_1))\}, \\ \{\{\neg s(w_2,x_2), q(w_2,x_2,y_2)\}, \\ \{\neg s(w_3,x_3), r(y_3,g(y_3))\}\} \end{array}$$

Um die Lesbarkeit noch zu erhöhen, erlauben wir daß dieselben Variablen in mehreren Klauseln auftauchen dürfen:

$$\begin{array}{l} \{p(w,h(w)), s(w,h(w))\}, \\ \{\{\neg s(w,x), q(w,x,y)\}, \\ \{\neg s(w,x), r(y,g(y)))\}\} \end{array}$$

Falls die Variablen für eine bestimmte Anwendung disjunkt sein müssen (z.B. für die Resolutionsregel), können wir jederzeit gebunden umbenennen. Man sollte nie vergessen, daß PL-Klauseln implizit allquantifiziert sind.

Kapitel 3

Die Tableaumethode

3.1 Informelle Einführung

Grob gesprochen, kann man die Tableaumethode als einen Widerspruchsbeweis durch Fallunterscheidung bezeichnen, wobei solange weitere Unterfälle generiert werden bis elementare Widersprüche erreicht sind. Wir sehen uns das an einem Beispiel an.

Beispiel 3.1 Man betrachte folgendes Problem aus der elementaren Mengenlehre. Die ersten vier Zeilen implizieren die letzte.

$$\begin{array}{lrcl}
(1) & S \cap Q & = & \emptyset \\
(2) & P & \subseteq & Q \cup R \\
(3) & P = \emptyset & \leadsto & Q \neq \emptyset \\
(4) & Q \cup R & \subseteq & S \\
\hline
(5) & P \cap R & \neq & \emptyset
\end{array}$$

Wir wollen einen Beweis durch Widerspruch führen, nehmen also an, daß

$$P \cap R = \emptyset \qquad \text{(Annahme)}$$

Wir beginnen den Beweis mit (3), das besagt daß entweder $P \neq \emptyset$ oder $Q \neq \emptyset$ gelten muß, was zur ersten Fallunterscheidung in Abbildung 3.1 führt. Seien also c bzw. d beliebige Elemente aus P bzw. Q. Wir führen nur den ersten Fall zu Ende, der Zweite geht analog.

Die Annahme besagt, daß kein Element zugleich in P und R sein kann, insbesondere gilt das für das oben eingeführte c, d.h. $c \notin P$ oder $c \notin R$. Da die erste Möglichkeit sofort zu einem Widerspruch führt, bleibt die zweite.

Zeile (2) besagt, daß jedes Element, das in P vorkommt, in Q oder in R vorkommen muß. Wieder wird spezialisiert zu: wenn $c \in P$, dann $c \in Q$ oder $c \in R$. Da wir schon wissen, daß $c \in P$ und $c \notin R$ bleibt nur $c \in Q$ übrig.

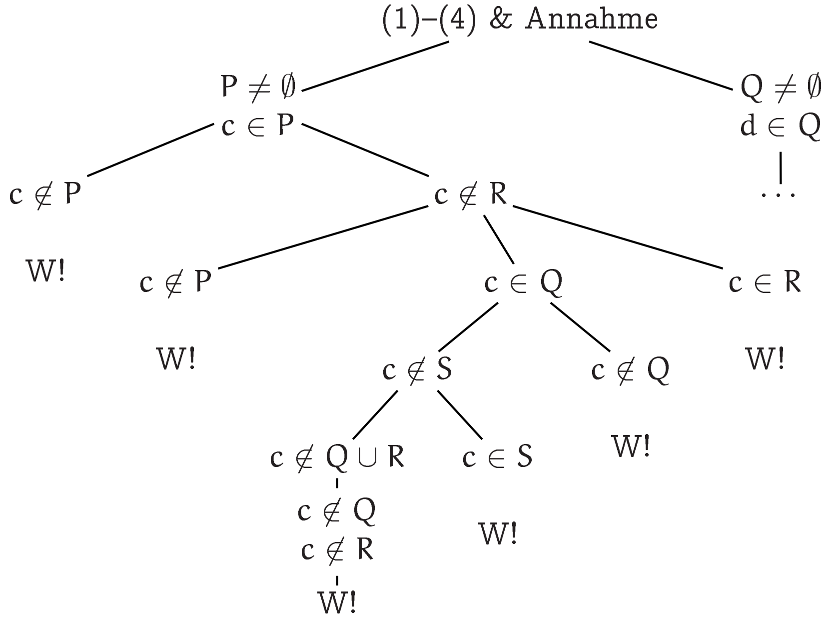


Abb. 3.1: Struktur des informellen Beweises aus Beispiel 3.1.

In gleicher Weise wie die Annahme wird (1) als $c \notin S$ oder $c \notin Q$ verwendet. Nur die erste Möglichkeit ist konsistent mit $c \in Q$.

Schließlich wird (4) spezialisiert zu: wenn $c \in Q \cup R$, dann $c \in S$. Dessen Kontraposition ergibt mit der schon abgeleiteten Information $c \notin S$, daß $c \notin Q \cup R$. Also kommt c weder in Q noch in R vor. Dies ist ein Widerspruch und da keine anderen Fälle übrigbleiben sind wir fertig.

In Abbildung 3.1 ist die Struktur des Beweises mit allen Unterfällen dargestellt. ■

Wir stellen fest, daß im Beispiel die notwendigen Fallunterscheidungen jeweils aus der syntaktischer Form der Annahme bzw. der Voraussetzungen generiert wurden. Für die Anwendung von Zwischenergebnissen bzw. Voraussetzungen war eine geeignete Instantiierung oder Spezialisierung notwendig. Die Struktur des Beweises ließ sich einfach als ein endlichverzweigender, mit Behauptungen markierter Baum darstellen. Eine Verzweigung im Baum entspricht dabei gerade einer Fallunterscheidung oder einer Disjunktion. Formalisiert man diese Beobachtungen, erhält man die Tableaumethode.

3.2 Automatisches Beweisen mit Tableaus

3.2.1 Grundtableaus für PL-Formeln ohne Normalform

Im folgenden wird davon ausgegangen, das bekannt ist, was Bäume sind. Eine mathematisch präzise Definition findet sich in Smullyan [1995].

$$\frac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta}{\beta_1 \mid \beta_2} \qquad \text{(Fallunterscheidung)}$$

$$\frac{\gamma}{\gamma_1(t)} \begin{array}{c} t \in Term^0_\Sigma \\ \text{beliebig}\end{array} \qquad \frac{\delta}{\delta_1(c)} \text{ c neu} \qquad \text{(Instantiierung)}$$

Tabelle 3.1: Erweiterungsregeln für Grundtableaus ohne Normalform.

Definition 3.2 *Ein* Tableau[1] *für eine endliche Formelmenge*

$$M = \{\phi_1, \ldots, \phi_m\} \in For_\Sigma$$

ist ein endlichverzweigender, markierter Baum, der wie folgt aufgebaut ist:

1. *Der lineare Baum mit genau* $|M|$ *Knoten* $\{\phi_1, \ldots, \phi_m\}$ *ist ein Tableau für* M.

2. *Ist* T *ein Tableau für* M, B *ein Zweig in* T *und* ϕ *eine Formel vom Typ* $\alpha, \ldots, \delta$ *auf* B, *dann erhält man wieder ein Tableau für* M, *indem man die in Frage kommende Regel aus Tabelle 3.1 auf* ϕ *anwendet und* B *mit den in der Konklusion entsprechenden Formeln erweitert.*

■

Die Zuordnung der Formeln in den Prämissen zu den Formeln in den Konklusionen in den Regelschemata entnehme man Tabelle 2.1. Anweisungen, wie man Formeln durch Fallunterscheidung analysiert, sowie für die Instantiierung (γ-Regel) und die Beschaffung von Beispielen (δ-Regel) können den Tabellen 3.1 und 2.1 entnommen werden. Die Erkennung von Widersprüchen ist im Begriff der Geschlossenheit eines Tableaus untergebracht.

Definition 3.3 *Ein Zweig in einem Tableau heißt* geschlossen, *falls er ein komplementäres Paar von Formeln enthält. Ein Tableau heißt* **geschlossen**, *falls jeder Zweig geschlossen ist.* ■

Satz 3.4 *Seien* $\psi \in For_\Sigma$ *und* $\Phi \subset For_\Sigma$ *abgeschlossen* (Φ *endlich*). *Es gibt ein geschlossenes Tableau für* $\Phi \cup \{\neg\psi\}$ *genau dann wenn* $\Phi \models \psi$.

[1] Der Plural von „Tableau“ ist im Deutschen „Tableau<u>s</u>“ und im Englischen „Tableau<u>x</u>“.

Beweis: Siehe Schmitt [1992a], Smullyan [1995] oder Fitting [1996]. Man beachte, daß der Satz Gebrauch von Proposition 2.32 macht. ■

Grundtableaus verdanken ihre Benennung der Tatsache, daß die Regeln (Instantiierung) gebundene Variablen durch Grundterme ersetzen. Dies führt auch dazu, daß auf Grundtableaus nur geschlossene Formeln vorkommen. Ihr Hauptnachteil besteht darin, daß die richtige Instanz, die zum Abschluß beiträgt, zum Zeitpunkt der Regelanwendung nicht bekannt ist. Eine systematische Aufzählung aller Grundterme über der Signatur eines gegebenen Problems führt zu extrem redundanten und viel zu großen Tableaus.

Beispiel 3.5 Das Eingangsbeispiel kann leicht formalisiert werden. Betrachte dazu die Signatur $\langle\{\}, \{p, q, r, s\}, \alpha\rangle$ mit $\alpha(p) = \alpha(q) = \alpha(r) = \alpha(s) = 1$.

Eine einstellige Relation p über einem nichtleeren Universum D ist einfach eine Teilmenge von D. $p(c)$ ist erfüllbar, falls $I(p)(c) = W$. Man kann also $p(x)$ als $x \in P$ interpretieren, wobei $P = \{c \in D \mid I(p)(c) = W\}$.

In nicht näher spezifizierten Interpretationen kann man Formeln über einstellige Prädikaten $p, q, \ldots$ daher als Aussagen über Mengen auffassen. Insbesondere entspricht die Konjunktion dem Durchschnitt von Mengen, die Disjunktion der Vereinigung und die Implikation entspricht der Teilmengenrelation. Existenzielle Quantifizierung bedeutet, daß die im Wirkungsbereich spezifizierte Menge nicht leer ist. Übersetzt man nach diesem Schema die Aussagen (1)–(4) und (Annahme) aus Beispiel 3.1, erhält man folgende Formeln:

(1)	$\neg\exists x(s(x) \wedge q(x))$
(2)	$\forall x(p(x) \to (q(x) \vee r(x)))$
(3)	$\neg\exists x(p(x)) \to \exists y(q(y))$
(4)	$\forall x((q(x) \vee r(x)) \to s(x))$
(Annahme)	$\neg\exists x(p(x) \wedge r(x))$

Ein partiell geschlossenes Grundtableau für diese Formelmenge ist in Abbildung 3.2 dargestellt. Die durchgezogenen Bögen geben die Abschlußpaare in den Zweigen an. Die geraden Linien repräsentieren die Baumstruktur des Tableaus. Der Zweig, der mit der schattierten Formel $q(d)$ endet, ist noch offen. Man beachte die Ähnlichkeit zu Abbildung 3.1. ■

3.2.2 Tableaus mit freien Variablen für Formeln erster Stufe ohne Normalform

Tableaus mit freien Variablen (FV-Tableaus) umgehen den Hauptnachteil der Grundtableaus dadurch, daß die γ-Regel nicht Grundterme „rät“, sondern freie

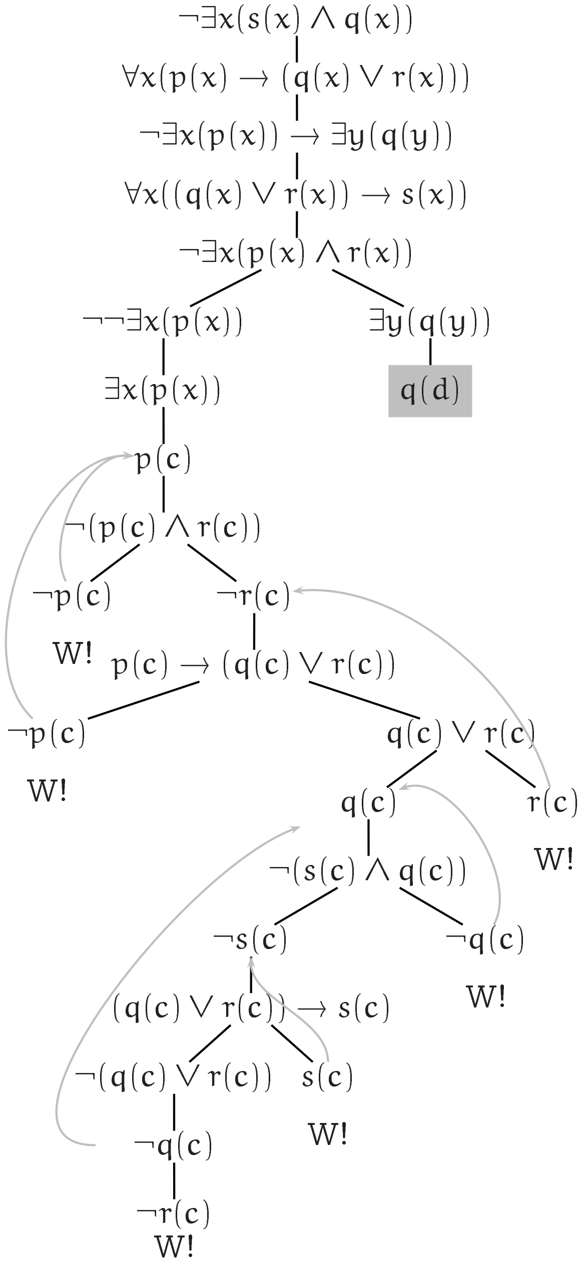


Abb. 3.2: Partieller Grundtableaubeweis der Formeln aus Beispiel 3.5.

$$\frac{\gamma}{\gamma_1(X)} \qquad \frac{\delta}{\delta_1(f_{[\delta]}(x_1,\ldots,x_n))}$$
$$X \text{ neu} \qquad \{x_1,\ldots,x_n\} = \mathrm{Frei}(\delta)$$

$$\begin{array}{c} \phi(s_1,\ldots,s_n) \\ \vdots \\ \psi(t_1,\ldots,t_n) \\ \hline\hline \sigma \end{array} \qquad \text{(Abschlußregel)}$$
$$\sigma \text{ mgu von } \{s_i, t_i\},\ 1 \le i \le n$$
$$\phi(s_1,\ldots,s_n)\sigma,\ \psi(t_1,\ldots,t_n)\sigma \text{ komplementär}$$

Tabelle 3.2: Erweiterungs- und Abschlußregeln für FV-Tableaus ohne Normalform.

Variablen erzeugt, die erst „bei Bedarf" durch Unifikation instanziiert werden, vgl. Reeves [1987]; Schmitt [1987]; Fitting [1990, 1996].

Die AL-Regeln des FV-Tableaukalküls stimmen überein mit den Regeln (Fallunterscheidung) in Tabelle 3.1, während die Regeln (Instantiierung) durch die in Tabelle 3.2 angegebenen Regeln ersetzt werden.

Satz 3.6 (Vollständigkeit von FV-Tableaus) *Seien $\psi \in For_\Sigma$, $\Phi \subset For_\Sigma$ abgeschlossen (Φ endlich). Wenn $\Phi \models \psi$, dann gibt es ein geschlossenes FV-Tableau für $\Phi \cup \{\neg\psi\}$.*

Beweisskizze: *Vollständigkeit:* Die Vollständigkeit der FV-Tableaus wird zum Beispiel in Schmitt [1992a]; Fitting [1996] bewiesen. Im übrigen folgt sie aus der Vollständigkeit der Grundtableaus, indem (Abschlußregel) durch die Regel

$$\frac{T}{T\sigma} \quad \sigma \text{ Grundsubstitution} \qquad \text{(Substitutionsregel)}$$

ersetzt wird, die nur genau einmal als letzte Regel angewandt werden darf. Denn sei ein geschlossenes Grundtableau T gegeben. Sei

$$\sigma = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$$

die Grundsubstitutionen, die den Anwendungen der γ-Regeln in T entspricht. Sei T' genau wie T aufgebaut, aber mit FV-Regeln. Dann ist $T'\sigma$ offenbar geschlossen.

Der Beweis der Vollständigkeit des unmodifizierten Kalküls erfordert ein sogenanntes Lifting-Lemma, das folgendes besagt:

> Zu jedem geschlossenen Grundtableau T' existiert ein FV-Tableau T mit geeigneten Anwendungen von (Abschlußregel) und zugehörigen Substitutionen $\sigma_1, \ldots, \sigma_m$, sowie eine Grundsubstitution ρ, so daß $T\sigma_1 \cdots \sigma_m \rho = T'$.

Wir werden in Abschnitt 4 solch ein Resultat für Klauseltableaus zeigen. ■

Der Beweis der Korrektheit von FV-Tableaus erfordert es, die Erfüllbarkeit von Tableaus zu definieren. Da ein Tableau freie Variablen enthalten kann, hat man zwei Möglichkeiten:

Definition 3.7 *Sei* T *ein FV-Tableau.* T *ist* erfüllbar*, falls es eine Interpretation* $\mathcal{D} = \langle D, I \rangle$ *gibt, so daß für alle Belegungen* β *auf wenigstens einem Zweig* B *von* T *alle Formeln von* $\mathcal{D}$ *und* β *erfüllt werden (Schreibweise:* $\mathcal{D}, \beta \models B$*). Falls* $\mathcal{D}$ T *erfüllt, schreiben wir* $\mathcal{D} \models T$. ■

In unserer Definition der Erfüllbarkeit von FV-Tableaus werden die in T vorkommenden freien Variablen als implizit allquantifiziert behandelt. Der Grund für diese Entscheidung ist zunächst nicht klar, zum Beispiel verwendet Fitting (1990) folgende Definition für die Erfüllbarkeit von FV-Tableaus:

> Sei T ein FV-Tableau. T ist erfüllbar, falls es für alle Grundsubstitutionen σ eine Interpretation $\mathcal{D}_\sigma = \langle D, I \rangle$ gibt, so daß $\mathcal{D}, \beta \models B$ für wenigstens einem Zweig B von $T\sigma$.

Die δ-Regel muß, wie alle Tableau-Regeln, die Erfüllbarkeit des Tableaus erhalten. Angenommen, T ist ein erfüllbares Tableau, das heißt, es gibt eine Interpretation $\mathcal{D} = \langle D, I \rangle$, so daß für alle Belegungen β für wenigstens einen Zweig $\mathcal{D}, \beta \models B$. Sei δ eine Formel auf B. Also gilt $\mathcal{D}, \beta_x^d \models \delta$ für ein geeignetes $d \in D$, und $\mathcal{D}$ kann wie in Satz 2.53 durch $I(f_{[\delta]}(\beta(x_1), \ldots, \beta(x_n))) = d$ zu einer Interpretation für $\delta_1(f_{[\delta]}(x_1, \ldots, x_n))$ erweitert werden.

Da $\{x, f(\ldots, x, \ldots)\}$ niemals unifizierbar sein kann, ist auch der kritische Wert $val_{\mathcal{D},\beta}(f(x_1, \ldots, x_n))$ für alle möglichen Substitutionen der $\{x_1, \ldots, x_n\}$ ausgeschlossen.

Es können aber weitere Variablen x frei auf B vorkommen, und deren Substitution darf durch die Erweiterung von $\mathcal{D}$ den Wahrheitswert von B nicht beeinflussen. Weil in Definition 3.7 alle $x \in Frei(B)$ implizit allquantifiziert sind, ist dies auch so. Würde dagegen Fittings Erfüllbarkeits-Begriff zugrundegelegt, könnte durch eine Variable, für die $\beta(x) = d' \neq d$ festgelegt ist, und eine Substitution $\{x \leftarrow f(x_1, \ldots, x_n)\}$ die Erfüllbarkeit von B verlorengehen. Die Erfüllbarkeit nach Fitting bleibt bei Anwendung unserer δ-Regel nicht immer erhalten. In Fitting [1990] ist dementsprechend auch nur die Korrektheit einer Variante unserer δ-Regel bewiesen, bei der alle freien Variablen auf dem Zweig als Argument der Skolemfunktion auftauchen.

Die Korrektheit unserer Variante wurde in Hähnle and Schmitt [1994] bewiesen, wobei die Skolemfunktion f neu auf dem Tableau sein mußte. Die Einsicht, daß stattdessen $f_{[\delta]}$ ausreicht stammt aus Beckert *et al.* [1993].

Satz 3.8 (Korrektheit von FV-Tableaus) *Seien* $\{\psi\}$ *und* $\Phi \subset For_\Sigma$ *abgeschlossen (*Φ *endlich). Wenn es ein geschlossenes FV-Tableau für die Formelmenge* $\Phi \cup \{\neg\psi\}$ *gibt, dann gilt* $\Phi \models \psi$*.*

Beweis: Geschlossene Tableaus sind unerfüllbar, da jeder Zweig komplementäre Paare enthält. Wir zeigen: Alle Tableaus für eine erfüllbare Formelmenge M sind erfüllbar. Daraus folgt dann, daß es für eine erfüllbare Formelmenge kein geschlossenes Tableau geben kann.

Sei also M eine erfüllbare, endliche, abgeschlossene Formelmenge (über Σ); und sei T ein Tableau für M. Also hat M nach Satz 2.53 ein kanonisches Term-Modell $\mathcal{D}^*$ (über Σ^*).

Wir zeigen per Induktion über die für die Konstruktion von T notwendige Anzahl von Regelanwendungen, daß $\mathcal{D}^*$ Modell eines jeden Tableaus für Φ ist, also auch von T.

Induktionsbehauptung: $\mathcal{D}^*$ erfüllt T, wenn nicht mehr als n Regelanwendungen zur Konstruktion von T notwendig sind.

Induktionsanfang: trivial.

Induktionsschritt: Zur Konstruktion von T' seien $n + 1$ Regelanwendungen notwendig, T' werde durch eine Regelanwendung aus T konstruiert. Die Induktionsbehauptung is auf T anwendbar.

Erweiterungsregel: Sei B ein Ast in T, und T' entstehe durch Regelanwendung auf ϕ und Erweiterung von B. Sei β beliebige Variablenbelegung.

Es gilt $\mathcal{D}^*, \beta \models B_0$ für einen Ast B_0 von T. Falls $B \neq B_0$ fertig, da dann B_0 Ast in T' und also $\mathcal{D}^*, \beta \models T'$. Andernfalls gilt $\mathcal{D}^*, \beta \models B$.

Die Formel ϕ ist von einem Typ $\alpha, \ldots, \delta$. Alle Fälle außer δ sind Routine bzw. wie in Fitting [1996]. Sei also $\phi = \delta$ mit $\mathrm{Frei}(\delta) = \{x_1, \ldots, x_n\}$. Sei B' der aus B entstandene Ast in T'.

Es reicht zu zeigen, daß $\mathcal{D}^*, \beta \models \delta_1(f_{[\delta]}(x_1, \ldots, x_n))$. Das folgt aber direkt aus dem Skolem-Lemma, da nach Voraussetzung $\mathcal{D}^*, \beta \models \delta$.

Abschlußregel: Sei nun T' aus T durch Anwendung der Abschlußregel mit Substitution σ entstanden, d.h. $T' = T\sigma$. Sei β' eine beliebige Variablenbelegung.

Sei die Variablenbelegung β definiert durch $\beta(x) = \mathrm{val}_{I,\beta'}(\sigma(x))$. Damit gilt $\mathrm{val}_{I,\beta'}(t\sigma) = \mathrm{val}_{I,\beta}(t)$.

Da $\mathcal{D}^*, \beta \models B$ für einen Ast B von T, gilt $\mathcal{D}^*, \beta' \models B\sigma$. $B' = B\sigma$ ist aber ein Ast in T' und also gilt

$\mathcal{D}^*, \beta' \models T'$. ∎

FV-Tableaus mit Unifikation wurden zwar erst Mitte der 80er Jahre eingeführt (in verschiedenen Varianten und unabhängig voneinander in Reeves [1987]; Schmitt [1987]; Fitting [1990]). Für speziellere Kalküle wurden jedoch ähnliche Regeln schon früher von Andrews (1981) und Bibel (1981) verwendet. Die Idee, freie Variablen, manchmal auch *dummy* Variablen genannt, zu verwenden, geht auf Prawitz (1960) und Kanger (1957) zurück. Diese hatten freilich noch keine Unifikation zur Verfügung, die erst von Robinson (1965b) eingeführt wurde.

Beispiel 3.9 In Abbildung 3.3 ist die FV-Version von Abbildung 3.2 dargestellt. Die gestrichelten Bögen markieren die Elternformeln, die durchgezogenen Bögen geben die Abschlußpaare in den Zweigen an. Die geraden Linien repräsentieren die Baumstruktur des Tableaus. Der Zweig, der mit der schattierten Formel $q(d)$ endet, ist noch offen. ∎

3.2.3 Wo ist die Prozedur indeterministisch?

Anhand des (partiellen) Tableaubeweises aus dem vorigen Unterabschnitt wollen wir die Stellen herausarbeiten, an denen eine Tableauprozedur Freiheitsgrade aufweist, also nicht deterministisch ist. Folgende vier Typen von Indeterminismus bei der Tableaukonstruktion lassen sich grundsätzlich unterscheiden.

1. Auswahl des nächsten für eine Erweiterung oder einen Abschluß zu betrachtenden Zweigs.

2. Soll versucht werden, den Zweig abzuschließen oder wird er vielmehr erweitert?

3. Falls abgeschlossen wird: mit welchen Formeln (es kann mehrere mögliche Abschlußpaare geben)?

4. Falls erweitert wird: mit welcher Formel?

Die verschiedenen Indeterminismen beleuchten das folgende fundamentale Problem:

Die Vollständigkeit der Tableauregeln garantiert nur die *Existenz* eines geschlossenen Tableaus. Eine systematische Tableaubeweisprozedur muß dafür sorgen, daß ein solches auch gefunden wird.

Man beachte, daß ein Abschluß eines Zweiges mit Hilfe einer Substitution erfolgen kann, die die schnelle Fertigstellung des Beweises verhindern kann. Ein einfaches Beispiel ist die Formel

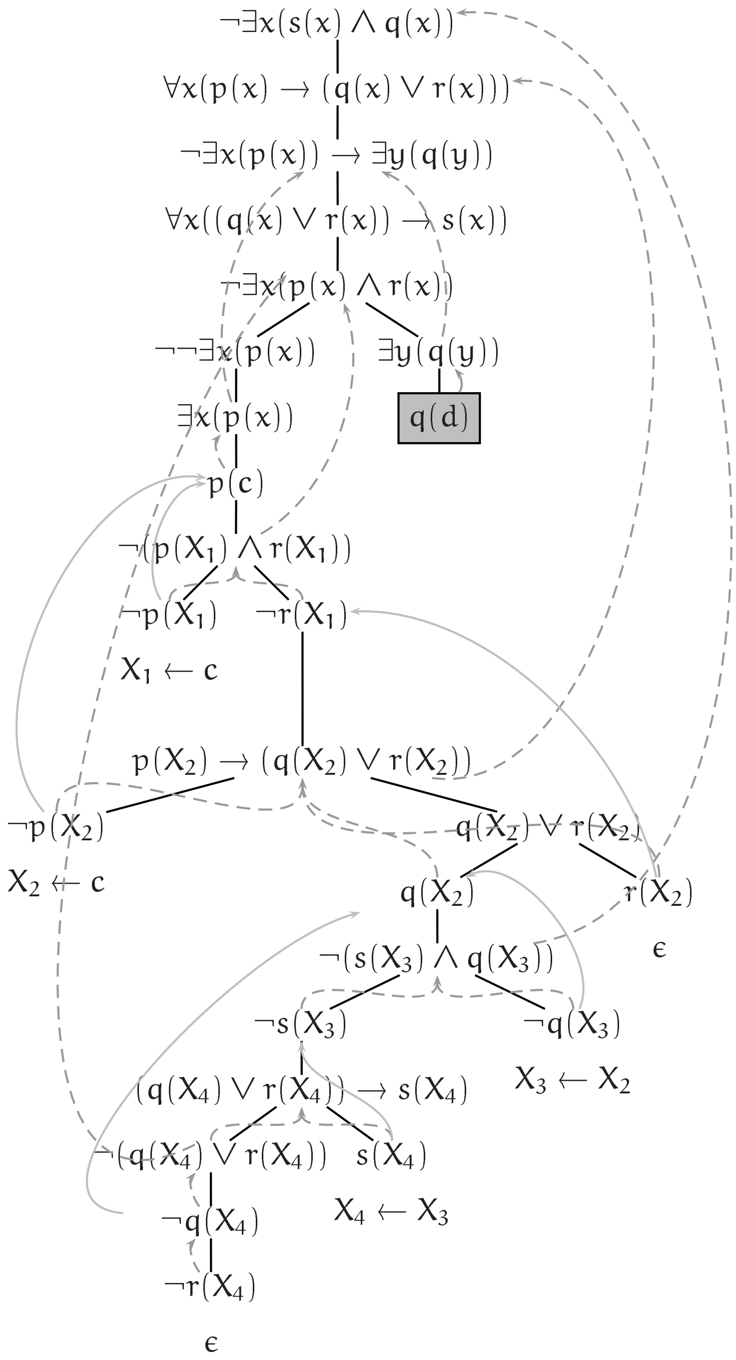


Abb. 3.3: Partieller FV-Tableaubeweis der Formeln aus Beispiel 3.5.

$$\neg(\forall x(p(x) \wedge q(x)) \to \exists y((p(y) \vee q(a)) \wedge p(b))) \ .$$

Ein Tableau hierfür beginnt wie folgt:

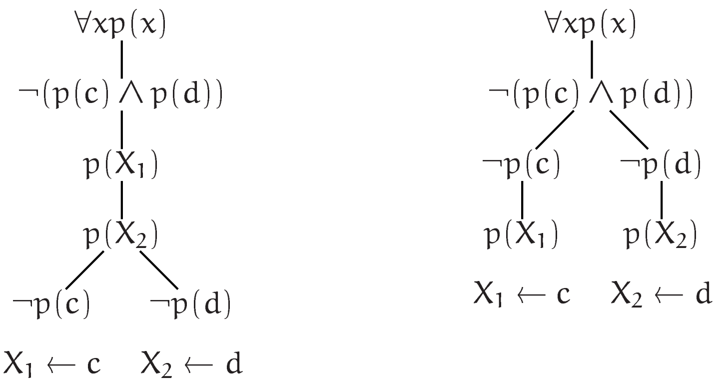
$$\begin{array}{ccc}
 & \forall x(p(x) \wedge q(x)) & \\
 & | & \\
 & \neg\exists y((p(y) \vee q(a)) \wedge p(b)) & \\
 & | & \\
 & p(X) & \\
 & | & \\
 & q(X) & \\
 & | & \\
 & \neg((p(Y) \vee q(a)) \wedge p(b)) & \\
 \swarrow & & \searrow \\
\neg(p(Y) \vee q(a)) & & \neg p(b) \\
| & & \\
\neg p(Y) & & \\
| & & \\
\neg q(a) & &
\end{array}$$

An dieser Stelle kann das Tableau mit der Substitution $\{X \leftarrow b, Y \leftarrow b\}$ geschlossen werden. Falls der rechte Zweig zuerst betrachtet wird oder im linken Zweig das Abschlußpaar $\{p(X), \neg p(Y)\}$ genommen wird, ist dies auch der Fall. Mit dem Abschlußpaar $\{q(X), \neg q(a)\}$ und der Substitution $\{X \leftarrow a\}$ kann der rechte Zweig jedoch nicht geschlossen werden. Es ist auch leicht, sich vorzustellen, daß die Formel $\neg p(Y)$ noch gar nicht auf dem Zweig vorhanden ist, wenn der Abschluß mit $\{q(X), \neg q(a)\}$ schon möglich ist (es reicht, im Beispiel $p(y)$ durch $(p(y) \vee r)$ zu ersetzen).

Wird der linke Zweig mit der „falschen" Substitution $\{X \leftarrow a\}$ geschlossen, muß $\forall x(p(x) \wedge q(x))$ nochmals im rechten Zweig verwendet werden. Dabei kann es natürlich wieder zu falschen Substitutionen kommen usw. Offensichtlich können auf diese Weise extrem redundante und damit große Tableaus selbst für einfache Probleme entstehen.

Eine Möglichkeit, um die Entstehung redundanter Tableaus zu verhindern, ist *alle* möglichen Tableaus aufzuzählen, denn darunter müssen auch kurze Beweise sein. Dies sollte allerdings nicht in naiver Weise geschehen, sondern mit Hilfe einer Reihe naheliegender Verbesserungen:

1. Verbesserung: nicht *alle* sondern nur „genug" Beweise, z.B. ist offenbar nur eines der folgenden beiden Tableaus notwendig:



2. Verbesserung: man kann die bisherigen Tableauregeln verstärken oder zusätzliche Regeln hinzufügen, um so möglicherweise kürzere Beweise zu finden. Allerdings erhöht sich dabei natürlich die Anzahl zu konstruierender Tableaus, so daß eine solche Vorgehensweise ein zweischneidiges Schwert ist.

3. Verbesserung: die Reihenfolge, in der die möglichen Tableaus aufgezählt werden, ist von großer Bedeutung für das Auffinden kurzer Beweise. Eine naheliegende Heuristik ist es, α-Formeln vor β-Formeln zu expandieren, da erstere nicht zu einer Verzweigung führen. Das Problem mit solchen und ähnlichen Heuristiken ist, daß immer auch Beispiele gefunden werden können, für die sie schlecht sind. Sie sind also nicht problemunabhängig und ohne eine genaue Beschreibung der Problemklasse für die sie nützlich sind wenig sinnvoll. Die systematische Generierung von Heuristiken aus eingegrenzten Problemklassen ist noch wenig untersucht. Wir werden auf solche *problemspezifischen* Verbesserungen nicht weiter eingehen können.

3.2.4 Eine abstrakte Tableauprozedur

Bevor wir genauer auf mögliche Verfeinerungen und Erweiterungen der Tableauprozedur zu sprechen kommen, gilt es zunächst, diese zu präzisieren. In Abbildung 3.4 und 3.5 ist eine Formulierung in Pseudo-Pascal angegeben.

Wie in Abbildung 3.4 zu sehen werden als Datenstrukturen für Zweige Listen verwendet, Tableaus sind Listen von Zweigen. Das Ergebnis einer Regelanwendung ist formal eine Liste von Listen von Formeln, also ein Tableau.

In konkreten Implementierungen sind einfache Listen als Datenstrukturen weniger geeignet, da der Zugriff und Änderung zu teuer ist. In effizienten Implementierungen verwendet man Haschtabellen. Die Listenrepräsentation hat für uns dennoch Vorteile:

1. viele Tableauvarianten lassen sich einfach und elegant damit beschreiben;

```
/* declare */

type branch = list of formula;
type tableau = list of branch;

var H: {extend,close};
var P,Q: formula;
var B,C: branch;
var T,U: tableau;

function read_next_formula(): formula;
function append(L,M: list):list;
function delete(X: elem; L: list):list;
function rest(L: list):list;
function first(L: list):elem;
function mgu(P,Q: formula): substitution;
function apply_subst(S: substitution; T: tableau): tableau;
function apply_rule(P: formula): tableau;

/* initialize */

B := [];
while ¬eof do
  B := append(B,[read_next_formula]);
T := [B];
```

Abb. 3.4: Deklarations- und Initialisierungsteil der Tableauprozedur.

2. im nächsten Abschnitt lernen wir eine sehr elegante und kompakte Implementierung kennen, die auf Listen basiert.

Das Hauptprogramm ist in Abbildung 3.5 angegeben. Geschlossene Zweige werden gelöscht. Also bedeutet `T=[]`, daß alle Zweige geschlossen sind. Unabhängig vom Modus wird im Rumpf der Schleife zunächst der ausgewählte Zweig gelöscht. Falls er geschlossen wird, ist man mit einem Durchlauf fertig, sonst werden die neuen Äste im letzten append an das Tableau angehängt. Das vorletzte append legt fest, daß auch in Zweigen das letzte Element das jüngste ist.

Die Implementierung der schattierten Funktionen ist offen; sie stellen die oben diskutierten Indeterminismen dar. In Abbildung 3.6 geben wir eine *mögliche* deterministische Implementierung dieser Funktionen an.

```
/* prove */

while T≠[] do begin
  B := select_branch(T);
  H := select_mode(B);
  T := delete(B,T);
  if H=close then begin
    [P,Q] := select_pair(B);
    T := apply_subst(mgu(P,Q),T)
    end
  else begin
    P := select_formula(B);
    if type(P) ∈ {alpha,beta,delta} then
      delete(P,B);
    U := apply_rule(P);
    while U≠[] do begin
      C := append(B,first(U));
      U := rest(U);
      T := append(T,[C])
    end
  end
end
writeln('Proof.')
```

Abb. 3.5: Hauptprogramm der Tableauprozedur.

```
function select_branch(T: tableau): branch;
begin
  return(first(T))
end;

function select_mode(B: branch):{close,extend};
begin
  if ∃σ:Pσ,Qσ komplementäres Paar then
    return(close)
  elseif B enthält Nichtliteral
    return(expand)
  else
    halt
end;

function select_formula(B: branch): formula;
begin
  P:= das erste Nichtliteral P ∈ B;
  return(P)
end;

function select_pair(B:branch):list of formula;
begin
  [P,Q] := die ersten P,Q ∈ B sodaß
           ∃σ:Pσ,Qσ komplementäres Paar;
  return([P,Q])
end;
```

Abb. 3.6: Mögliche Implementierung der Funktionen, die den Indeterminismus der Tableauprozedur darstellen.

`select_branch`	Neue Zweige werden hinten angehängt, also wird eine Schlange implementiert.
`select_mode`	Abschluß wird immer vorgezogen, wenn er möglich ist: „greedy“ Implementierung. Diese Führt für AL zu Terminierung.
`select_formula`	Neue Formeln werden hinten angehängt, also `depth_last`.
`select_pair`	Am weitesten oben stehendes Paar.

Die so spezifizierte Tableauprozedur würde beispielsweise für die Eingabe $\neg(p(c) \wedge p(d)), \forall x p(x)$ einen Beweis konstruieren, der dem rechten Tableau auf Seite 39 entspricht. Die Implementierung von `select_branch` stellt eine Schlange dar, da neue Zweige hinten angehängt werden. `select_mode` versucht sobald wie möglich abzuschließen.

Die Tableauprozedur ist nicht für alle Implementierungen der Auswahlfunktionen vollständig. Zum Beispiel würde die Eingabe $\forall x p(x), \neg(p(c) \wedge p(d))$ wegen der Implementierung von `select_formula` zur wiederholten Epandierung der ersten Formel führen, ohne daß die zweite Formel je genommen wird. Probleme dieser Art werden im nächsten Unterabschnitt besprochen.

Um die Vollständigkeit einer konkreten Tableauprozedur zu zeigen, reicht es nachzuweisen, daß für alle widersprüchlichen Formelmengen ein geschlossenes Tableau nach Definition 3.2 existiert, das Instanz eines mit der Tableauprozedur erzeugten Tableaus für diese Formelmenge ist. Dies kann im Einzelfall technisch sehr aufwendig werden.

3.2.5 Bevorzugungsprobleme (Fairneß)

In diesem Unterabschnitt stellen wir unvollständige Tableauprozeduren dar, bei denen die Unvollständigkeit durch die unzulässige Bevorzugung bestimmter Aktionen vor anderen zustandekommt.

Bevorzugung eines bestimmten Literals für den Abschluß – select_pair: Es wird immer wieder dasselbe Literal für den Abschluß genommen. Dadurch wird der Abschluß der übrigen Äste verhindert (Abbildung 3.7).

Bevorzugung einer bestimmten Formel – select_formula: Es wird immer wieder dieselbe Formel expandiert (Abbildung 3.8). Es kann auch das Beispiel auf Seite 39 zur Illustration genommen werden.

Bevorzugung eines Modus – select_mode: Entgegen einem eventuellen ersten Eindruck, ist auch die vorgestellte Implementierung von `select_mode`

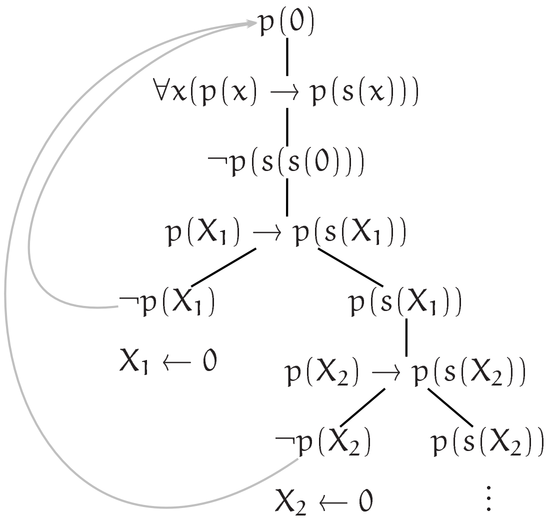


Abb. 3.7: Unvollständigkeit durch Bevorzugung eines bestimmten Literals für den Abschluß.

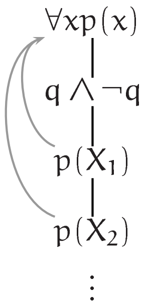


Abb. 3.8: Unvollständigkeit durch Wiederholte Expansion derselben Formel.

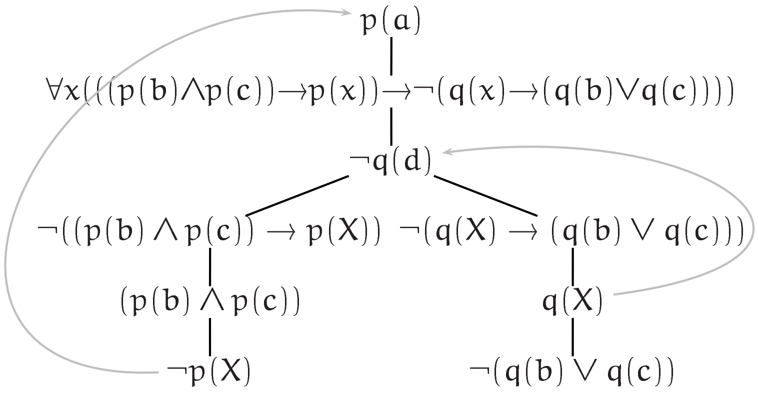


Abb. 3.9: Unvollständigkeit durch Bevorzugung von Abschlüssen.

(greedy) problematisch wie man in Abbildung 3.9 sieht. Der Grund ist, daß sich die Tableaumethode an der Formelstruktur orientiert. Die komplementären Paare, die zur Lösung führen, stehen im Beispiel erst zur Verfügung nachdem die erste Abschlußmöglichkeit ignoriert wurde. Schreibt man die Formel als Matrix (wobei Disjunktionen senkrecht, Konjunktionen waagerecht geschrieben werden) wird dies offenbar:

$$\begin{bmatrix} p(a) & \begin{bmatrix} [[\ p(b) & p(c)\ ]\neg p(X)] \\ [[\neg q(b) & \neg q(c)]\ q(X)\ ] \end{bmatrix} & \neg q(d) \end{bmatrix}$$

Falls `select_mode` greedy ist, wird die gestrichelten Konnektion verhindert.

`select_branch` erzeugt *kein* Bevorzugungsproblem, weil alle Zweige geschlossen werden müssen. Im Gegensatz zu den übrigen Entscheidungen, die ODER-Knoten in einem Suchbaum darstellen, ist die Zweigauswahl ein UND-Knoten. Andererseits kann die Zweigauswahlstrategie die Gestalt und Größe des gefundenen Beweises stark beeinflussen.

Im folgenden wollen wir verschiedene Lösungen des Bevorzugungsproblems ansprechen.

Breitensuche Die Implementierung der Auswahlfunktionen (das heißt, der Prozeduren `select_literal`, `select_branch` und `select_formula`) stellen kritische Alternativen dar. Wir stellen uns die Suche nach einem Tableaubeweis als einen UND-ODER-Suchbaum vor, wobei die möglichen Werte dieser Funktionen die ODER-Knoten definieren. In Abbildung 3.11 ist dies für das schon oben verwendete Zählerbeispiel ausgeführt. Die durchgezogenen Linien stellen weitere Alternativen dar, die zunächst nicht betrachtet werden. Die gestrichelten Linien sind die untersuchte Alternative, die zum Erfolg führt.

Eine vollständige Breitensuche ist wegen des Speicherbedarfs außer für kleine Probleme kaum möglich.

Lösung des Bevorzugungsproblems durch faire Implementierung Eine weitere Lösung ergibt sich durch Umgehung des Indetermismus in `select_mode` und `select_pair` (siehe Abbildung 3.10). `select_mode` testet das gesamte Tableau auf Abschluß. Nur falls es ein σ gibt, das ganz T schliesst, wird *überhaupt* ein Zweig geschlossen. Solch eine schliessende Substitution σ wird zurückgegeben und dann in `select_pair` verwendet. Es werden also alle Schritte der Tableaukonstruktion – bis auf den letzten – nicht-destruktiv gemacht.

Von `select_pair` werden also nur Zweigabschlüsse, die mit σ kompatibel sind, gewählt. Alle Abschlußregeln kommen nach allen Erweiterungsregeln.

Definition 3.10 *`select_formula` heißt* **fair** *falls für alle $\Phi \subset For_\Sigma$ jede Folge von Tableaus $(T_i)_{i\geq 0}$ für Φ folgende Eigenschaften hat:*

```
function select_mode(T: tableau):
                     [{close,extend},substitution];
begin
  if ∃ Substitution σ so daß Tσ geschlossen then
    return ([close,σ])
  elseif T enthält Nichtliteral
    return ([extend,ε])
  else
    halt
end;

function select_pair(B:branch, σ:substitution):
                     list of formula;
begin
  [P,Q] := die ersten P,Q∈B so daß Pσ,Qσ komplementäres Paar;
  return ([P,Q])
end;
```

Abb. 3.10: Faire Implementierung von select_pair und select_mode.

1. *Auf alle Nichtliterale in allen Zweigen aller* T_i*, wird eine passende Regel angewandt*

2. *Für jedes* $k \geq 0$ *gibt es ein* i_k*, so daß auf alle* γ*-Formeln in allen Zweigen von* T_{i_k} *mindestens* k*-mal eine* γ*-Regel angewandt wird.*

■

Fairneß der Formelauswahl bewirkt, daß jede universelle in jedem Zweig in beliebig vielen Kopien zur Verfügung steht und daß jede komplexe Formel irgendwann analysiert wird. Damit kann eine Situation wie im Beispiel am Anfang dieses Abschnitts nicht entstehen.

Nachteile der Lösung mit Fairneß

- Das gesamte Tableau muß gespeichert werden, da das ganze Tableau auf einmal abgeschlossen wird und Substitutionen sich global auswirken.
- Das Testen auf Abschluß ist entsprechend teuer, da bei jedem Erweiterungsschritt alle Zweige auf Abschluß getestet werden, auch solche, die schon vorher betrachtet wurden, was sehr ineffizient ist.

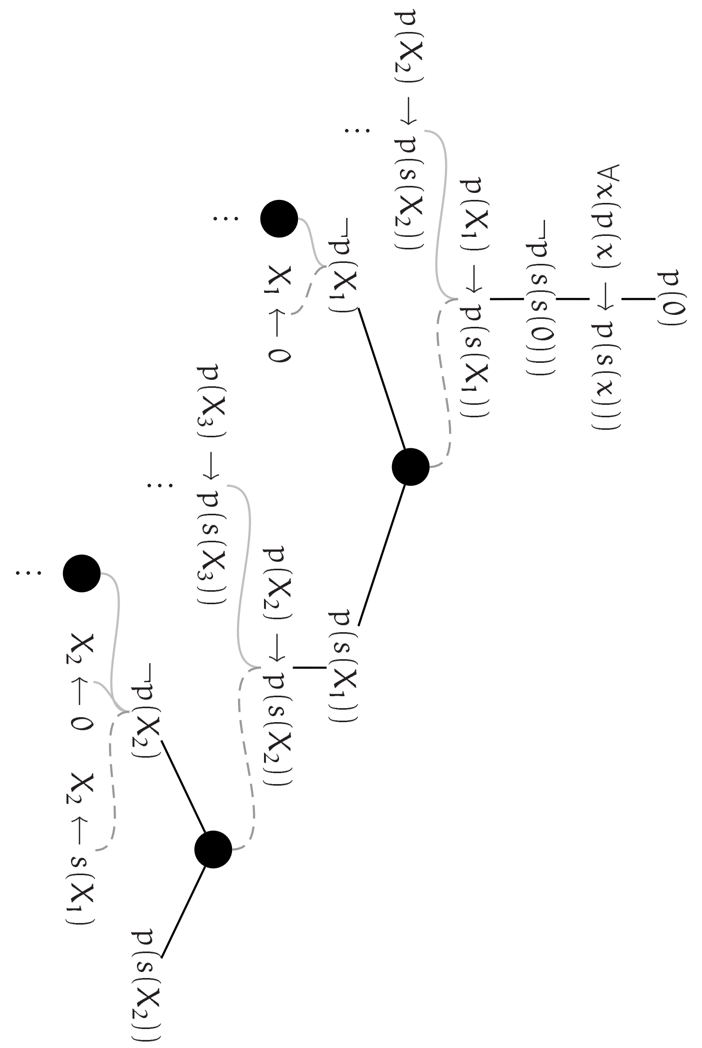


Abb. 3.11: Anfang des vollständigen Suchbaums für das Zählerbeispiel.

- Die Implementierung ist nicht ganz einfach, sie erfolgt durch Warteschlangen oder globale Zähler an Unterformeln. Die Vollständigkeit ist schwierig einzusehen, wenn noch andere Heuristiken (Gewichtung) zur Formelauswahl hinzukommen.

- Durch Ausmultiplizieren wird das Tableau schnell zu groß. Zum Beispiel erzeugen je zwei Kopien der zwei Formeln

$$\forall x(p(x) \vee q(x) \vee r(x)),\ \forall x(s(x) \vee t(x) \vee u(x))$$

  $3^2 3^2 = 3^4 = 81$ Zweige und mehr als 120 Knoten.

Vorteile der Lösung mit Fairneß Ein großer Vorteil der fairen Implementierung ist folgende Eigenschaft:

Definition 3.11 *Ein Kalkül heißt* **beweiskonfluent**, *falls jeder angefangene Beweis für eine unerfüllbare Formelmenge vervollständigt werden kann.* ■

Man beachte, daß im Gegensatz zur Konfluenz von Termersetzungssystemen hier nicht gefordert wird, daß angefangene Beweise zu *demselben* Beweis vervollständigt werden können.

Satz 3.12 *FV-Tableaus sind beweiskonfluent, falls (Abschlußregel) in eine einzige Anwendung von (Substitutionsregel) verschoben wird.*

Beweisskizze: Sei T ein beliebiges geschlossenes Tableau für ϕ, B ein beliebiger Zweig in T, der durch $\langle L, \overline{L}\rangle$ geschlossen sei. Weiterhin sei T' ein beliebiges Tableau für ϕ, B' ein beliebiger offener Zweig in T'.

Durch endlich ist ein Tableau unterhalb von B' zu konstruieren, in dem in *jedem* Zweig $\langle L, \overline{L}\rangle$ auftaucht (Details: Übung).

Da B' beliebig war, erhält man so eine Erweiterung T'' von T', in der jeder Zweig geschlossen werden kann. T' wird also im wesentlichen ignoriert und seine offenen Zweige werden mit Kopien von T geschlossen. ■

Beweiskonfluente Kalküle erfordern keine Breitensuche und auch kein Backtracking, das im nächsten Abschnitt vorgeschlagen wird.

Ein weiterer Vorteil ist die Verträglichkeit mit sogenannten *Redundanz*- oder *Deletion*-Strategien, die wir später bei Resolutionskalkülen kennenlernen werden. Deren Grundidee ist, Kriterien anzugeben unter denen ein aus T entstandenes Tableau T'

- Hilfe von T zu vereinfacht werden kann oder

- zu zeigen, daß T' redundant ist und nicht betrachtet werden braucht.

Die erheblichen Nachteile haben eine ernsthafte Implementierung des Fairneß-basierten Ansatzes bislang verhindert. Zumindest wird eine faire Formelauswahl aber oft als Heuristik bei der folgenden Lösung mit Backtracking zu verwendet: eine Priorisierung der fairen Alternativen in `select_formula` ordnet den Suchraum evtl. günstiger an.

Ideal wäre es, die Fairneß auf Substitutionen auszudehnen, das heißt, daß in jedem Zweig jede Typ γ Formel nach und nach in allen benötigten Instanzen erzeugt wird. Anders gesagt, geht es darum `select_pair` fair zu implementieren.

In Verbindung mit inkrementeller Tiefensuche (wie im nächsten Abschnitt beschrieben) ließe sich dann auf die Speicherung des gesamten Tableaus verzichten und doch (Abschlußregel) verwenden. Es ist jedoch ein offenes Problem, eine konkrete und praktikable faire Implementierung von `select_pair` anzugeben.

Lösung des Bevorzugungsproblems durch Backtracking Alle ernsthaften Tableaubeweiser arbeiten mit impliziter Beweisaufzählung durch Backtracking:

Wann immer `select_xxx` indeterministisch ist, erzeugt man einen *Choicepoint*.

Da es ∞ viele Tableaus gibt, muß systematisch in einer immer größeren Menge von Tableaus gesucht werden, falls man Breitensuche vermeiden will, was das Konzept des Vollständigkeitsmodus nahelegt.

Definition 3.13 *Ein* Vollständigkeitsmodus m *ist eine monotone Abbildung von* $\mathbb{N}$ *auf endliche Mengen von Tableaus, so daß* $\bigcup_{i \in \mathbb{N}} m(i)$ *alle Tableaus umfaßt.* ■

Beispiele für Vollständigkeitsmodi:

- Tableaus mit höchstens i Knoten.
- Tableaus, in denen γ-Regel höchstens i-mal auf eine Formel im Tableau/in jedem Zweig angewandt wird.
- Tableaus mit Zweigen höchstens der Länge i.
- Tableaus, in denen höchstens Terme der Länge bzw. Tiefe i vorkommen.
- Kombinierte Maße.

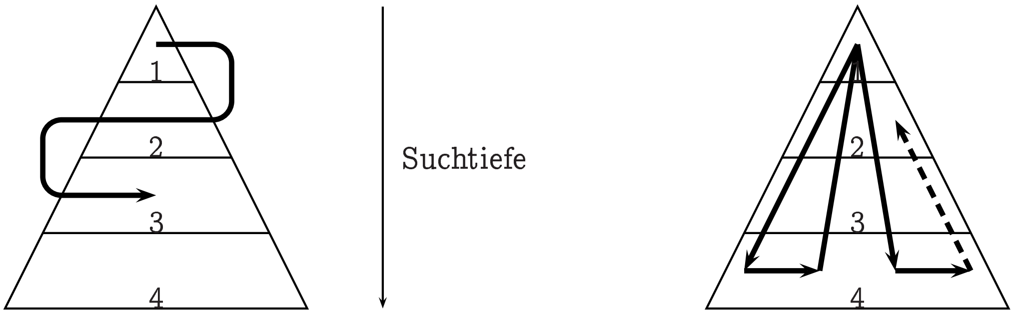


Abb. 3.12: Breitensuche (links) und beschränkte Tiefensuche (rechts).

$M(i)$ sei der Teil des UND-ODER Suchbaums, der Tableaus die in $m(i)$ aber nicht solche in $m(j)$, $j < i$ umfaßt. Breitensuche rät z.B. i_0 und durchsucht $M(i_0)$.

Inkrementelle Tiefensuche[2] Stickel [1992], durchsucht $\bigcup_{j<i} M(i)$ für $i = 0, 1, \ldots$ jeweils mit Tiefensuche und impliziter Aufzählung durch Backtracking.

Da $|m(i)|$ exponentiell wächst ist ein adäquates Maß wichtig. Außerdem ist zu beachten:

- Genau analysieren, welche Choicepoints für die Vollständigkeit benötigt werden. Nur die wirklich benötigten werden behalten.
- Die Reihenfolge der Choicepoints kann großen Einfluß auf den Suchraum haben.

Die inkrementelle Tiefensuche vereinigt Vorteile der Tiefen- und der Breitensuche: bei der inkrementellen Tiefensuche werden zwar die Tableaus in $m(i)$ auch in allen $m(j)$ mit $i < j$ durchsucht, aber da die Größe der $m(i)$ exponentiell wächst, ist der Gesamtaufwand im Vergleich zu einer Breitensuche mit der richtigen Tiefe nicht wesentlich höher und der Platzbedarf exponentiell kleiner. Außerdem braucht man die richtige (das heißt minimale, für die es ein geschlossenes Tableau gibt) Tiefe nicht zu kennen.

Während unbeschränkte Breitensuche (in Abbildung 3.12 links) auf der Ebene i die Speicherung aller Tableaus in $m(i)$ erfordert, genügt es bei beschränkter Tiefensuche mit Backtracking (in Abbildung 3.12 rechts) die Tableaus zu speichern, die auf einem vertikalen Pfad durch den Suchraum liegen.

Für die Tiefensuche werden Sprachkonstrukte der dynamischen Programmierung (Backtracking) eingeführt (Abbildung 3.13).

Für die inkrementelle Tiefensuche werden nun an geeigneten Stellen der Tableauprozedur `fail` Kommandos eingefügt (Übung: implementieren Sie die oben aufgeführten Vollständigkeitsmodi).

In den indeterministischen Funktionen werden den Alternativen entsprechende Choicepoints erzeugt (Abbildung 3.14)/

[2] Engl.: depth-first iterative deepening search.

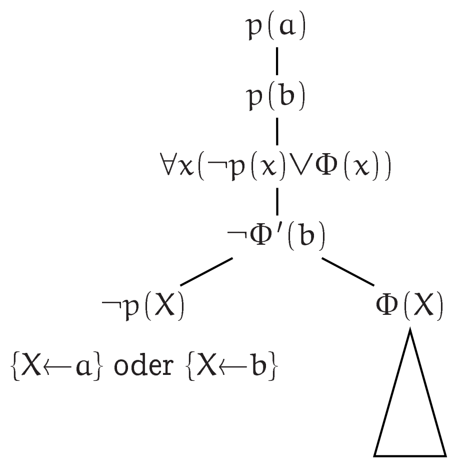
```
function or(L: list of X):X;
begin
  if L=[] then
    fail
  else
    lege choicepoint an mit der Code-Fortsetzung:
      'L := rest(L); return(or(L));'
  return(first(L))
end;

procedure fail;
begin
  if es gibt noch Choicepoints then
    gehe zum zuletzt angelegten Choicepoint
  else
    halt
end;
```

Abb. 3.13: Die Funktionen `or` und `fail` zur Implementierung von Backtracking.

Die Reihenfolge der Elemente in den Listen `L` kann erhebliche Auswirkungen auf die Größe des zuerst gefundenen Beweises haben.

Wir geben ein Beispiel für diesen Phänomen anhand von `select_pair`; als Übung möge man sich ein Beispiel überlegen, bei dem die Formelauswahl kritisch ist (Hinweis: modifiziere die Formelauswahl in dem Tableau aus Abbildung 3.3).



Wird der linke Zweig zuerst mit der Substitution $\{X \leftarrow a\}$ geschlossen, kann der rechte Zweig nicht mehr geschlossen werden. Falls Φ eine komplexe Formel ist, kann es lange dauern, bis Backtracking zur richtigen Substitution $\{X \leftarrow b\}$ im linken Zweig zurückführt.

Aus dem Satz über Beweiskonfluenz folgt, daß bei *fairer* Implementierung

```
function select_mode(B: branch):{close,extend};
begin
  if ∃σ:Pσ,Qσ komplementäres Paar then
    return (or([close,expand]))
  elseif B enthält Nichtliteral
    return (expand)
  else
    fail
end;

function select_formula(B: branch): formula;
begin
  L:= Liste aller Nichtliterale P ∈ B;
  return (or(L))
end;

function select_pair(B:branch): list of formula;
begin
  L := Liste aller Paare P,Q ∈ B so daß ∃σ:Pσ,Qσ komplementäres Paar;
  return (or(L))
end;
```

Abb. 3.14: Indeterministische Versionenen der Auswahlfunktionen.

die Funktion `select_formula` ohne Schaden für die Vollständigkeit deterministisch gemacht werden kann, falls alle Substitutionsmöglichkeiten mittels der Choicepoints in `select_pair` und `select_mode` aufgezählt werden. Der geringere Suchraum kann den Nachteil längerer Beweise leicht aufwiegen.

Aus den gleichen Gründen kann auch die Funktion `select_branch`, auch *Auswahlregel* genannt, deterministisch gemacht werden. Die spezielle Implementierung von `select_branch` kann jedoch unterschiedliches Verhalten der Tableauprozedur hervorrufen, da sie mit den in `select_pair` bestimmten Substitutionen interagiert; es gibt zwar keinen Einfluß auf die Vollständigkeit, wohl aber auch den Suchraum. Dazu folgendes Beispiel:

$$\begin{array}{c} p(a) \\ | \\ p(b) \\ | \\ q(a) \\ | \\ \forall x(\neg(p(x) \wedge q(x))) \\ | \\ \neg(p(X) \wedge q(X)) \\ / \quad \backslash \\ \neg p(X) \quad \neg q(X) \end{array}$$

Wird der linke Zweig zuerst ausgewählt hat man dort zwei Alternativen zum Abschluß. Nur eine führt zum Abschluß des Tableaus. Im rechten Zweig dagegen ist der Abschluß determiniert.

Hier noch einige übliche Auswahlregeln:

Least recent entspricht unserer Beispielprozedur, falls das letzte Listenelement in `select_branch` genommen wird.

Left-first, right-first „linkester/rechtester" offener Zweig.

Depth-first: ein offener Zweig mit maximaler Länge.

Depth-last: ein offener Zweig mit minimaler Länge.

Depth-left-first: der „linkeste" offene Zweig maximaler Länge.

3.2.6 Zwei Optimierungen

Links

Folgende Einschränkung verhindert, daß Formeln ausgewählt werden, die mit dem bisherigen Beweis nichts zu tun haben.

Definition 3.14 *Sei T ein Tableau. Ein Paar Formeln $\langle \phi, \psi \rangle$ auf demselben Zweig $B \in T$ ist ein* Link, *wenn es Unterformeln ϕ', ψ' von ϕ, ψ mit entgegengesetzter Polarität gibt, sowie eine Substitution σ, die $\langle \phi', \psi' \rangle$ zu einem komplementären Paar macht. ($P = Q$ ist dabei zugelassen).* ■

Offenbar ist es sinnlos, in `select_formula` eine Formel zu wählen, die kein Link zu einer Formel (und sei es auf sich selbst) auf dem aktuellen Zweig besitzt.

Beispiel: $\forall x((q(x) \vee r(x)) \rightarrow s(x))$ und $\neg \exists x(p(x)) \rightarrow \exists y(q(y))$ sind ein Link (via $q(x)$ und $q(y)$ im initialen Zweig), nicht aber $\neg \exists x(s(x) \wedge q(x))$ und $\neg \exists x(p(x) \wedge r(x))$.

Als Verstärkung der Link-Bedingung kann man zusätzlich die skolemisierte Form von ϕ und ψ betrachten.

Im nächsten Kapitel werden wir häufig mit Links arbeiten.

Universelle Formeln

Die folgende Optimierung macht bestimmte Substitutionen unnötig und ermöglicht dadurch kürzere Beweise.

Definition 3.15 *Sei P eine Formel auf einem Ast B eines Tableaus T. Das Tableau T' entstehe dadurch, daß $\forall x P$ zu B hinzugefügt wird.*[3]

Die Formel P heißt universell *auf B bzgl. x, falls*

$$T \models T' \ ,$$

wobei T und T' als Formeln mit den Disjunktionen ihrer Äste identifiziert werden, und diese wiederum mit der Konjunktion ihrer Formeln. ■

Eine Formel ist also (bzgl. einer bestimmten Variablen) universell, wenn sie im aktuellen Kontext generalisiert werden kann.

Lemma 3.16 *Eine Formel P auf einem Tableauzweig B ist insbesondere dann universell bzgl. einer Variablen x (auf B), wenn für alle Grundsubstitutionen σ gilt: $B\sigma \models (\forall x P)\sigma$.*

Beim Abschluß kann die erforderliche Substitution nun auf nicht-universelle Variablen eingeschränkt werden. Beispiel:

$$\forall x p(x)$$
$$\neg p(a) \vee \neg p(b)$$
$$p(X)$$
$$\neg p(a) \qquad \neg p(b)$$
$$\epsilon \qquad \epsilon$$

[3] In der Regel ist x eine der Variablen in P; das muß aber nicht so sein.

Hier ist jeweils (auf beiden Ästen) $p(X)$ universell bzgl. X. Ohne die Optimierung muß eine zweite Instanz von $\forall x p(x)$ erzeugt werden.

Universalität ist im allgemeinen unentscheidbar, aber es gibt ein starkes hinreichendes Kriterium:

Satz 3.17 *Eine Formel P auf einem Tableauast B ist universell bzgl. x, falls*

1. *P aus der Anwendung einer γ-Regel entstand, wobei x die neu eingeführte freie Variable ist,*
2. *P aus der Anwendung einer γ-, δ- oder α-Regel auf eine Formel, die bzgl. x universell ist, entstand, oder*
3. *P aus der Anwendung einer β-Regel auf eine Formel, die bzgl. x universell ist, entstand und keine der anderen bei dieser Regelanwendung neu entstandenen Formeln die Variable x enthält (diese also nicht auf verschiedene Äste „verteilt“ wird).*

3.3 Implementierung der Tableaumethode in Prolog

3.3.1 Interpretation von Prolog-Programmen

In diesem Abschnitt stellen wir eine komplette, lauffähige Implementierung von FV-Tableaus mit folgenden Eigenschaften vor:

- inkrementelle Tiefensuche mit Backtracking
- Backtracking über `select_mode`, `select_pair`
- `select_formula` deterministisch & fair
- `select_branch` left-first
- (Abschlußregel) liefert mgu
- expliziter Zugriff nur auf aktuellen Zweig
- Eingabe ist geschlossene Formel in NNF ohne $\exists$

Zentrale Elemente der Implementierung der Tableauprozedur mit inkrementeller beschränkter Tiefensuche waren:

- Unifikation

- Backtracking

Beides wird von der Programmiersprache Prolog entscheidend unterstützt. Zur Theorie von Prolog kann Lloyd [1987]; Schmitt [1992b] empfohlen werden, zur Programmiertechnik in Prolog siehe Sterling and Shapiro [1986]; O'Keefe [1990]. Hier können wir nur eine ganz kurze Wiederholung der wichtigsten Grundlagen von Prolog geben.

Definition 3.18 Hornklauseln *sind Klauseln, die höchstens ein positives Literal enthalten.* Definite Hornklauseln *sind Klauseln, die genau ein positives Literal enthalten. Eine* negative Klausel *ist eine Klausel, die kein positives Literal enthält.*

Ein Prolog Programm *ist eine abgeschlossene SNF Formel über definiten Hornklauseln.* ■

Hornklauseln können mit $val_{\mathcal{D},\beta}$ wie ganz normale Formeln interpretiert werden. Man spricht dann von ihrer *deklarativen Semantik.* Aufgrund ihrer Eigenschaften kann man definite Hornklauseln jedoch auch als Implikationen oder Regeln auffassen und wie ein Programm abarbeiten. Dies wird als *operationale Semantik* bezeichnet. Um den operationalen Aspekt zu betonen, schreibt man

$$\{p, \neg p_1, \ldots, \neg p_n\} \qquad \text{bzw.} \qquad \{\neg p_1, \ldots, \neg p_n\}$$

als

```
p :- p1, ... ,pn.     bzw.     | ?- p1, ... ,pn.
```

oder als

$$p \leftarrow p_1, \ldots, p_n \qquad \text{bzw.} \qquad \leftarrow p_1, \ldots, p_n \;.$$

Die wichtigste Datenstruktur in Prolog sind Listen. Notation hierfür:

`[a,b,c]`, `[a|R]`

`[a|R]` entspricht dabei `push(a,R)` in üblicher Schreibweise. Wenn `R = []` ist das Ergebnis `[a]`. Weitere Datenstrukturen sind Zahlen und Zeichenketten, die im folgenden aber keine Rolle spielen.

Variablen werden groß, Prädikaten-, Konstanten- und Funktionssymbole werden klein geschrieben.

Beispiel für ein einfaches Prolog Programm.

```
append([],L,L).
append([E1|R1],L,[E1|R2]) :- append(R1,L,R2).
```

Dieses Programm hängt zwei Listen aneinander. Es ist nicht die effizienteste Lösung, die möglich ist (mehr dazu unter dem Stichwort „Differenzlisten“ und „Akkumulator“ in den oben genannten Büchern).

Eine Anfrage an ein Prolog Programm ist eine negative Hornklausel, z.B.:

```
| ?- append([a,b],[c,d],X).

X = [a,b,c,d]
```

Interpretation von Prolog Programmen. Die Interpretation von Anfragen über Prolog Programmen erfolgt durch die sogenannte SLD-Resolution (*L*inear Resolution with *S*election function for *D*efinite clauses), vergleiche auch Abschnitt 8.3.5.

Die Darstellung der Berechnung eines Prolog Programmes erfolgt durch Beweissuchbäume.

Definition 3.19 *Ein* Beweissuchbaum *ist ein endlich verzweigender Baum, dessen Knoten mit Prolog-Anfragen und dessen Kanten mit Substitutionen markiert sind. Jeder endliche Zweig ist entweder mit* fail *oder mit dem Symbol* $\Box$ *und einer Substitution markiert.*

Sei P *ein Prolog Programm,* G *eine Anfrage. Dann ist ein Beweissuchbaum für* $P \cup \{G\}$ *(und left_first Auswahlfunktion) wie folgt definiert:*

1. *Die Wurzel ist mit* G *markiert.*

2. *Für jeden Knoten mit Markierung* $\leftarrow p, p_1, \ldots, p_k$ *(*$k \geq 0$*), jede Regel* $q \leftarrow q_1, \ldots, q_n \in P$ *(*$n \geq 0$*), so daß das* ***Unterziel*** p *und* q *mit mgu* σ *unifizierbar sind, existiert genau ein Nachfolgerknoten mit Markierung* $(\leftarrow q_1, \ldots, q_n, p_1, \ldots, p_k)\sigma$.

   Falls $k = n = 0$*, erhält der Nachfolgerknoten Markierung* $\Box$.

3. *Mit* $\Box$ *markierte Knoten sind Blätter; ihr Zweig wird mit Vereinigung aller mgus oberhalb* $\Box$ *markiert.*

4. *Knoten ohne Nachfolger sind Blätter; ihr Zweig wird mit* `fail` *markiert.*

■

Das Prolog Programm und die Anfrage append von oben ergibt folgenden Beweissuchbaum:

```
←append([a,b],[c,d],X)
                |
                | X = [a|R2]
                |
←append([b],[c,d],R2)
                |
                | R2 = [b|R2']
                |
←append([],[c,d],R2')
                |
                | R2' = [c,d]
                |
                □
```

$$\{X \leftarrow [a,b,c,d],\ R2 \leftarrow [b,c,d],\ R2' \leftarrow [c,d]\}$$

Ein spezielles Literal „!" (**Cut**) in einer Regel bewirkt folgendes: Nach Abarbeitung von $\leftarrow !, p_1, \ldots, p_k$ werden alle noch nicht abgearbeiteten Zweige unterhalb des Unterziels, welches die den Cut einführende Regel aktivierte, abgeschnitten.

Beweissuchbäume sind ODER-Bäume; jeder erfolgreiche Zweig entspricht einer korrekten Antwort auf die Anfrage. Durch den Cut können also manche Antworten abgeschnitten werden.

Beispiel 3.20 `p(X) :- q(X),r.`

```
p(d).
q(X) :- s(X),!,t(X).
q(a).
s(b).
s(c).
t(c).
r.
| ?- p(X).
```

Der Beweissuchbaum für dieses Programm mit Anfrage ist in Abbildung 3.15 dargestellt. Wie dort ersichtlich ist, liefert die Anfrage die Antwort $\{X \leftarrow d\}$. Da der Cut in der ersten Regel für q vorkommt, werden alle Zweige unterhalb von q(X) abgeschnitten. Wird der Cut vor s(X) gelegt, wird die Antwort $\{X \leftarrow c\}$ gefunden, bevor das Unterziel, das den Cut enthält, beendet ist. Ohne Cut liefert das Programm alle drei Antworten. ∎

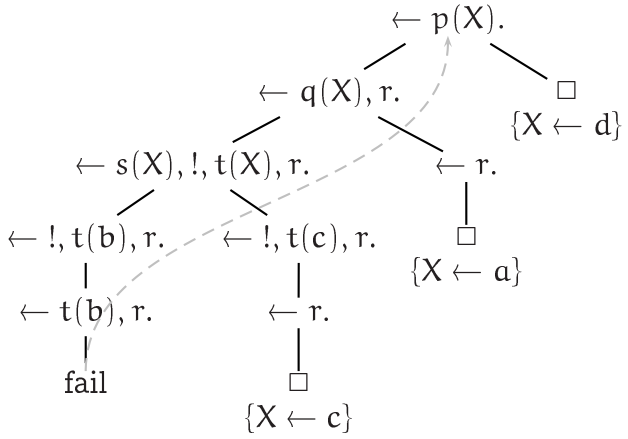


Abb. 3.15: Prolog Beweissuchbaum für Beispiel 3.20.

Unifikation in Prolog. *Unifikation* erfolgt während der Prolog-Interpretation an zwei Stellen:

Explizite Unifikation: mit `=` (ohne Occur-Check) oder `unify` (mit Occur-Check, eventuell langsamer). Der Occur-Check bedeutet, daß sichergestellt wird, daß keine zu belegende Variable auf der rechten Seite ihres Substitutionsterms vorkommt. Dies würde zu zyklischen Termen führen. Unterschiede ergeben sich zum Beispiel bei

```
| ?- p(X,X) = p(Y,f(Y)).
```

und

```
| ?- unify(p(X,X),p(Y,f(Y))).
```

Das Beispiel verlangt `X = Y` und `X = f(Y)`, woraus mit Transitivität die zyklische Substitution `Y = f(Y)` ensteht, die nur durch den unendlichen Term $\mathrm{f}(\mathrm{f}\cdots\mathrm{f}(\mathrm{Y})\cdots)$ gelöst wird.

Implizite Unifikation: (Ohne Occur-Check) jeweils zwischen Unterziel und Kopf einer Regel.

Wird die Unifikation zum Abschluß eines Tableauzweigs eingesetzt, so ist der Occur-Check für die Korrektheit der Tableauprozedur notwendig (Übung: man überlege sich ein entsprechendes Beispiel).

Negation in Prolog. Der Negations-Operator „`\+`“ ist in Prolog mit Negation-as-failure implementiert, d. h., die leere Substitution ist Antwort für `\+goal`, falls die Suche nach einer Antwort für `goal` fehlschlägt. `\+goal` verhält sich so, als wäre es mit

```
\+ goal :- goal, !, fail.
\+ goal.
```

implementiert.

If-then-else in Prolog. Es gibt in Prolog einen If-then-else-Operator der Form „`p -> q ; r`“ (if p then q else r). Er enthält einen lokalen Schnitt, der Backtracking über die Bedingung `p` verhindert, nachdem die Behandlung von `q` bzw. `r` begonnen wurde. „`p -> q ; r`“ entspricht `goal`, wenn `goal` wie folgt implementiert ist:

```
goal :- p, !, q.
goal :- r.
```

Choicepoints in Prolog. Choicepoints (also Verzweigungen im Beweissuchbaum) werden erzeugt, wenn

- Köpfe von verschiedenen Regeln mit einem Unterziel unifizierbar sind;
- durch den „`;/2`“ Operator in einer Regel („oder“); dies führt zu einer Verzweigung zwischen den Argumenten dieses Operators im Beweissuchbaum.

3.3.2 leanTAP: Das Programm

leanTAP wurde in Beckert and Posegga [1995][4] beschrieben. Das vollständige Programmlisting ist:

```
prove((A,B),UnEx,Lits,FreeV,Lim) :- !,
   prove(A,[B|UnEx],Lits,FreeV,Lim).

prove((A;B),UnEx,Lits,FreeV,Lim) :- !,
   prove(A,UnEx,Lits,FreeV,Lim),
   prove(B,UnEx,Lits,FreeV,Lim).

prove(all(X,Fml),UnEx,Lits,FreeV,Lim) :- !,
   \+ length(FreeV,Lim),
   copy_term((X,Fml,FreeV),(X1,Fml1,FreeV)),
   append(UnEx,[all(X,Fml)],UnEx1),
   prove(Fml1,UnEx1,Lits,[X1|FreeV],Lim).
```

[4] Dieses und weitere Papiere zu leanTAP und seinen Verwandten, sowie das Programm samt einem Testrahmen und Beispielen sind im World Wide Web unter der folgenden Adresse erhältlich: `http://i12www.ira.uka.de/~posegga/leantap/leantap.html`.

```
prove(Lit,_,[L|Lits],_,_) :-
   (Lit = -Neg; -Lit = Neg) ->
   (unify(Neg,L); prove(Lit,[],Lits,_,_)).

prove(Lit,[Next|UnEx],Lits,FreeV,Lim) :-
   prove(Next,UnEx,[Lit|Lits],FreeV,Lim).
```

3.3.3 leanTAP: Die Idee

Vorteilhafte Nutzung von Prolog:

- Backtracking
- Indexing
- Datenstrukturen (Terme)
- Unifikation

Die Anfrage

$$\texttt{prove(}\overbrace{\texttt{Fml,UnEx,Lits}}^{\text{Aktueller Zweig}}\texttt{,FreeV,Lim)}$$

ist für eine gegebene geschlossene Formel `Fml` in NNF ohne $\exists$ erfolgreich, wenn es ein geschlossenes Tableau für `Fml` mit höchstens `Lim` freien Variablen pro Zweig gibt.

Spezifische Designentscheidungen:

- Inkrementelle Tiefensuche mit Backtracking.
- Vollständigkeitsmodus m, wobei in $m(i)$ alle Tableaus mit höchstens i freien Variablen pro Ast enthalten sind.
- Expliziter Zugriff nur auf den aktuellen Ast; andere noch offene Äste und Choice-points werden (effizient) von Prolog verwaltet (Structure-sharing von Termen, Kopieren der Variablenbindungen anstatt ganzer Terme).
- Darstellung des aktuellen Astes:

`Fml`	nächste zu betrachtende Formel
`UnEx`	weitere noch zu betrachtende Formeln
`Lits`	schon betrachtete Literale

  – Separate Verwaltung von `Fml` im ersten Argument, da PROLOG-Regeln in Hashtabelle verwaltet werden mit Prädikat- und erstem Funktionssymbol des Kopfliterals als Schlüssel;
  – Trennung in `UnEx` und `Lits` vermeidet wiederholte Suche nach Nichtliteral bzw. Literal.

- Eingabe in NNF ohne $\exists$ (Regeln nur für $\wedge$, $\vee$, $\forall$).

Formelsyntax

Konjunktion `(A,B)` Infix

Disjunktion `(A;B)` Infix

Negation `-A` Präfix

Allquantor `all(X,A)` Präfix

Die Formel aus Beispiel 3.5 wird wie folgt transformiert:

```
(all(Y,(-(s(Y));-(q(Y)))),
 all(Z,(-(p(Z));q(Z);r(Z))),
 (p(c);q(d)),
 all(W,(-(q(W)),-(r(W));s(W))),
 all(X,(-(p(X));-(r(X)))))
```

Die Eingabeformel braucht nicht bereinigt zu sein, sie muß aber geschlossen sein. Die Eingabe von Formelmengen erfolgt als Konjunktion ihrer Elemente.

Die Anfrage

```
| ?- prove(Fml,[],[],[],Lim).
```

ist erfolgreich, wenn es ein geschlossenes Tableau für `Fml` in $m(\mathrm{Lim})$ gibt.

3.3.4 Konjunktion & Disjunktion

$\texttt{prove(}\overbrace{\texttt{Fml,UnEx,Lits}}^{\text{Aktueller Zweig}}\texttt{,FreeV,Lim)}$

$$\boxed{\dfrac{A \wedge B}{\begin{array}{c}A\\B\end{array}}}$$

```
prove((A,B),UnEx,Lits,FreeV,Lim):- !,
  prove(A,[B|UnEx],Lits,FreeV,Lim).
```

$$\boxed{\dfrac{A \vee B}{A \mid B}}$$

```
prove((A;B),UnEx,Lits,FreeV,Lim):- !,
  prove(A,UnEx,Lits,FreeV,Lim),
  prove(B,UnEx,Lits,FreeV,Lim).
```

Die jüngste Formel wird das erste Element von `UnEx`. Es werden keine Choicepoints erzeugt. Der Schnitt bewirkt, daß zusammengesetzte Formeln nicht auf Abschluß getestet werden.

In der Disjunktionsregel wird der linke Zweig immer zuerst geschlossen. Also ist eine `left_first` Auswahlregel implementiert.

3.3.5 Universelle Quantifizierung

$$\texttt{prove(}\overbrace{\texttt{Fml,UnEx,Lits}}^{\text{Aktueller Zweig}}\texttt{,FreeV,Lim)}$$

$$\boxed{\dfrac{\forall X\ \ Fml(X)}{Fml(X)}}$$

```
prove(all(X,Fml),UnEx,Lits,FreeV,Lim):- !,
 \+length(FreeV,Lim),
 copy_term((X,Fml,FreeV),(X1,Fml1,FreeV)),
 append(UnEx,[all(X,Fml)],UnEx1),
 prove(Fml1,UnEx1,Lits,[X1|FreeV],Lim).
```

Das spezielle Prädikat `\+` steht für die Negation (genauer: das Ergebnis des Arguments Erfolg/Nichterfolg wird vertauscht; dabei sollte das Argument beim Aufruf instanziiert sein). Falls die Länge der Liste `FreeV` der freien Variablen im Zweig das Limit `Lim` erreicht hat, wird also ein fail erzeugt.

Die spezielle Verwendung von `copy_term` ersetzt genau `X` in `Fml` durch eine neue Variable `X1` in `Fml1` und benennt die gebundenen Variablen um. Hier wird die Voraussetzung, daß die Ausgangsformel geschlossen sei, gebraucht.

Die Angabe von `FreeV` verhindert, daß die freien Variablen in `Fml` durch neue Variablen ersetzt werden, was `copy_term` sonst tun würde. Die Ersetzung anderer gebundener Variablen ist unkritisch, da dies gebundener Umbenennung entspricht. Dies ist der Hauptgrund für die Anwesenheit des Parameters `FreeV`.

Die benutzte γ-Formel wird an das *Ende* von `UnEx` getan. Dies entspricht einer *fairen* Implementierung von `select_formula`, weil die persistenten γ-Formeln somit in einer Schlange organisiert sind. Fairneß ist notwendig, da `select_formula` deterministisch implementiert ist.

3.3.6 Abschluß und Erweiterung von Ästen

$$\texttt{prove(}\overbrace{\texttt{Fml,UnEx,Lits}}^{\text{Aktueller Zweig}}\texttt{,FreeV,Lim)}$$

Abschluß:

```
prove(Lit,_,[L|Lits],_,_) :-
  (Lit = -Neg; -Lit = Neg) ->
    (unify(Neg,L)
    ;
     prove(Lit,[],Lits,_,_)).
```

Die Regel für den Abschluß ist die einzige, die nichtdeterministisch ist. Sie ruft sich rekursiv auf, bis ein komplementäres Paar gefunden ist. Die Angabe von '`[]`' im Aufruf verhindert, daß bei der Rekursion die fünfte Regel angesteuert wird. Die ersten drei Regeln können ebenfalls nicht angesteuert werden, da `Lit` ein Literal ist.

Die Prolog-Version des If-then-else enthält einen impliziten Schnitt; dies verhindert den Fall `--P = -Q`, der für `Lit = -p` durch Backtracking auftreten würde.

Die Bedingung des If-then-else erzeugt das Komplement `Neg` von `Lit`. Dann wird geprüft, ob es einen mgu von `Neg` und `L` gibt. Hierzu ist die Unifikation mit Occur-check notwendig, während zur Berechnung des Komplements die eingebaute Unifikation genügt.

Man beachte die Ähnlichkeit zur Implementierung von `select_pair` auf Seite 53.

Erweiterung:

```
prove(Lit,[Next|UnEx],Lits,FreeV,Lim):-
 prove(Next,UnEx,[Lit|Lits],FreeV,Lim).
```

Der Indeterminismus von `select_mode` wird dadurch verwirklicht, daß die letzte Regel (die Erweiterungsregel) erst gewählt werden kann, wenn alle Abschlüsse fehlgeschlagen sind.

Die aktuelle Formel muß zu diesem Zeitpunkt ein Literal sein, das zu `Lits` getan wird.

Dann wird das erste Element von `UnEx` zur aktuellen Formel, das heißt, daß `select_formula` deterministisch ist und die jüngste Formel zuerst expandiert wird.

leanTAP hat also gerade die Alternativen in `select_pair` und `select_mode` als Choicepoints; dagegen sind `select_branch` und `select_formula` deterministisch (im wesentlichen: depth-left-first.

Die Anordnung der γ-Formeln als Schlange ergibt eine faire Implementierung von `select_formula`, wodurch leanTAP vollständig ist.

Um eine unerfüllbare Formel zu widerlegen, muß leanTAP sukzessiv mit wachsenden Schranken `Lim=0,1,2,...` aufgerufen werden, was inkrementeller Tiefensuche entspricht.

leanTAP ist unter jedem DEC-10 Prolog lauffähig (z.B.: Quintus, SICStus, Eclipse); ggf. müssen Bibliotheken für `append`, `length`, `unify` und `copy_term` geladen werden.

Das Eingangsbeispiel kann in ca. 5 msec mit `Lim = 6` auf einer Sun Sparc 10/20 unter Sicstus Prolog 2.1.9 bewiesen werden.

Kapitel 4

Klauseltableaus

In diesem Kapitel beschränken wir die Eingabe für Tableauprozeduren auf universelle Klauseln erster Stufe, also Formeln in SNF. Daraus ergeben sich eine Reihe von Vorteilen für die Weiterentwicklung von Tableauprozeduren:

1. Die Tableauprozedur wird einfacher und uniformer; es lassen sich Verfeinerungen und Verstärkungen aufdecken und definieren, die im allgemeinen Fall zu erheblichen technischen Schwierigkeiten führen würden.
2. Der Vollständigkeitsbeweis wird ebenfalls weniger technisch und bleibt damit für kompliziertere Verfeinerungen handhabbar.
3. Es lassen sich leichter Vergleiche zu anderen klauselbasierten Beweisprozeduren wie Resolution führen.
4. Es ist eine effiziente Implementierung durch Erweiterung eines PROLOG Compilers möglich.

Natürlich hat die Verwendung von SNF auch Nachteile:

1. Die Verständlichkeit der Tableaubeweise leidet. Selbst unter Verwendung von definitorischer Normalformberechnung ist eine Rückübersetzung in volle Logik nicht immer möglich.
2. Manche nichtklassische Logiken besitzen keine SNF-Darstellung.

4.1 Tableauprozedur für SNF

Klauseln sind abgeschlossene Formeln in SNF. Schreibweise:

$$\{\{p(x), \neg q(y)\}, \{\neg p(x)\}\}$$

steht für

$$(\forall x \forall y (p(x) \lor \neg q(y))) \land (\forall z \neg p(z)) \ .$$

Eine Klausel der Form $\{l\}$ nennt man *Einerklausel* oder (anglisierend) Unitklausel.

In der folgenden Tableauprozedur gibt es nur eine einzige Erweiterungsregel, die man als eine Kombination der seitherigen γ-, β- und Substitutionsregel ansehen kann. Schematisch sieht diese wie folgt aus:

$$\frac{\overbrace{\forall x_1, \ldots, x_m (l_1 \vee \cdots \vee l_r)}^{C \in S}}{l_1\nu \mid \quad \cdots \quad \mid l_r\nu} \qquad \nu \text{ Variablenumbenennung}$$

Im Gegensatz zu früher ist die Startklauselmenge S nicht mehr Teil des initialen Tableaus.

Definition 4.1 *Ein* Klauseltableau *für eine (endliche) prädikatenlogische Klauselmenge* $S = \{C_1, \ldots, C_m\} \in For_\Sigma$ *ist ein endlichverzweigender, mit Literalen markierter Baum, der wie folgt aufgebaut ist:*

1. *Der Baum mit genau einem Knoten markiert mit* $\top$ *ist ein Klauseltableau für* S.

2. *Ist* T *ein Klauseltableau für* S, B *ein Zweig in* T, $C = \{l_1, \ldots, l_n\}$ *eine Klausel in* S, *und* ν *eine Variablenumbenennung, dann erhält man wieder ein Klauseltableau für* S, *indem* B *durch* n *neue Nachfolger erweitert wird mit Markierungen* $l_1\nu, \ldots, l_n\nu$.

3. *(Substitutionsregel) Sei* T *Klauseltableau für* S, σ *Substitution. Dann ist* $T\sigma$ *Klauseltableau für* S.

■

Der Abschluß ist wie früher definiert. Man erhält sofort eine Definition für Aussagenlogik durch Weglassen von ν und Regel 3. Die Korrektheit ergibt sich sofort als Spezialfall der Korrektheit von Nichtklauseltableaus.

Beispiel 4.2 Die Formelmenge aus Beispiel 3.5 in SNF ist

$$S' = \left\{ \begin{array}{l} \{\neg s(x), \neg q(x)\}, \{\neg p(x), q(x), r(x)\}, \{p(c), q(d)\}, \\ \{\neg q(x), s(x)\}, \{\neg p(x), \neg r(x)\}\} \end{array} \right.$$

Der Tableaubeweis in Abbildung 3.3 in der Variante für Klauseltableaus ist in Abbildung 4.1 angegeben. Die Pfeile deuten die Partner für die Abschlüsse an. ■

In Abbildung 4.2 ist eine Klauseltableauprozedur angegeben. Die Funktion `extend` (Abbildung 4.3) wird einmal bei der Initialisierung und einmal in der Schleife aufgerufen. Sie implementiert die Erweiterung eines Zweigs `B` mit einer

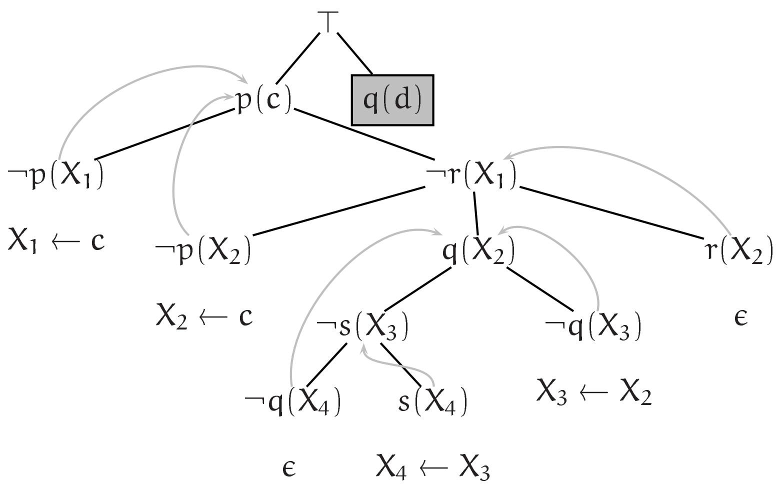


Abb. 4.1: Partielles Klauseltableau für die Klauselmenge aus Beispiel 4.2.

Klausel aus der Klauselmenge S. Entsprechend ist ein Zweig nicht eine Liste von Literalen sondern von Einerklauseln. `extend` gibt eine Liste mit den neugebildeten Zweigen zurück, formal ein Tableau. Die Erweiterung von Zweigen und die Anwendung von Substitutionen ist wie in der deklarativen Definition 4.1 getrennt. Variablenumbenennungen werden von `new_instance_of` ausgeführt.

Der folgende Satz garantiert, daß es für Vollständigkeitsbetrachtungen ausreicht, die deklarative Beschreibung von Klauseltableaus zu betrachten. Er besagt, daß obige Prozedur ein Klauseltableau konstruieren kann, wenn eines gemäß der deklarativen Beschreibung existiert. Das ist nicht selbstverständlich, da in der Prozedur (Abschlußregel) anstatt (Substitutionsregel) verwendet wird, d.h., Substitutionen ausschließlich mgus sind und die Reihenfolge ihrer Anwendung von `select_branch` abhängt. (Das Theorem ist trivialer Weise richtig, falls `select_mode` $\equiv$ `extend`.)

Satz 4.3 ***(Lifting)***
Sei T' *ein beliebiges geschlossenes Klauseltableau.*
Die Klauseltableauprozedur kann

(a) für eine geeignete Substitution σ

(b) bei geeigneter Implementierung von `select_clause`, `select_mode` *und* `select_pair` *oder mit Hilfe von Backtracking*

ein geschlossenen Klauseltableau T *erzeugen, so daß*

$$T' = T\sigma$$

Zum Beweis brauchen zunächst folgendes Lemma:

```
/* declare */

type clause;  /* universally quantified */
              /* disjunction of literals */
type instance = list of literal;
type input =  list of clause;
type branch = list of literal;
type tableau = list of branch;

var P,Q: literal;
var S: input;
var B: branch;
var T: tableau;
var σ: substitution;

/* initialize */

S := [];
while ¬eof do
  S := append(S,[read_next_clause]);

/* clausal tableaux */

T := [[]];
while T≠[] do begin
  B := select_branch(T);
  T := delete(B,T);
  if select_mode(B) = close then begin
    [P,Q] := select_pair(B);
    σ := mgu(P,Q);
    T := apply_subst(σ,T)
    end
  else
    T := append(T,extend(B,S))
end
writeln('Proof.')
```

Abb. 4.2: Eine Klauseltableauprozedur.

```
function extend(B:branch,S:input):tableau;
var U: tableau;
var C: clause;
var I: instance;
begin
  C := select_clause(B,S);
  I := new_instance_of(C);
  U := [];
  while I≠[] do begin
   U := append(U,append(B,[first(I)]));
   I := rest(I)
  end;
  return(U)
end;
```

Abb. 4.3: Die von der Klauseltableauprozedur in Abbildung 4.2 verwendete Funktion extend.

Lemma 4.4 *Sei T ein Tableau und τ eine Substitution, so daß T geschlossen ist. Der Ast $B\tau$ von $T\tau$ sei durch den Link $\langle L\tau, L'\tau\rangle$ geschlossen. ρ sei Mgu von $\langle L, L'\rangle$.*

Dann gilt $T\rho\tau = T\tau$.

Beweis: (Lemma) ρ ist ein Mgu, also existiert τ' mit $\tau' \circ \rho = \tau$. O.b.d.A. ist ρ idempotent, daher
$T\rho\tau = T\rho\rho\tau' = T\rho\tau' = T\tau$. ∎

Nun zum Beweis des Lifting-Satzes:

Beweis: (Lifting-Satz)
Sei T'' wie T' aber ohne Substitutionen; und sei τ die Substitution, so daß $T''\tau = T'$.

select_clause, select_mode seien so gewählt, daß Zweig B_1 von T'' erzeugt wird.

Da T' geschlossen ist, enthält $B_1\tau$ komplementäres Paar $\langle L_1\tau, L_1'\tau\rangle$.

Sei ρ_1 Mgu von $\langle L_1, L_1'\rangle$, welches zudem Ergebnis von select_pair sei.

Sei $T_1 = T''\rho_1$ erzeugt durch Anwendung der Abschlußregel.

Aus dem Lemma folgt: $T_1\tau = T''\rho_1\tau = T''\tau = T'$.

Induktion über die Äste von T'' liefert ein $T = T_n$, das von der Prozedur gefunden wird, so daß $T\tau = T'$ und T geschlossen. ∎

Mit dem Satz über Lifting kann man die Vollständigkeit der Klauseltableau*prozedur* für PL auf die Vollständigkeit von AL-Klauseltableaus reduzieren.

Satz 4.5 *Sei* S *eine unerfüllbare PL-Klauselmenge über* Σ. *Dann findet die Prozedur ein geschlossenes Klauseltableau* T *für* S.

Beweis:

1. Nach dem Satz von Herbrand 2.40 gibt es eine endliche Menge $S' \subset \mathrm{Term}^0_\Sigma(S)$ von Grundinstanzen von S, die unerfüllbar ist.

   Diese ist AL-unerfüllbar über der Signatur
   $$\Sigma' = \{p \mid p \text{ ist atomare Unterformel von } S'\} \ ,$$
   denn angenommen, S' hätte ein AL-Σ'-Modell I'. Dann ist I ein Σ-Modell für S' mit der Festlegung:
   $$I(p)(t) = \begin{cases} I'(p(t)) & t \in \mathrm{Term}^0_\Sigma \\ F & \text{sonst} \end{cases}$$
   Dabei machen wir uns zunutze, daß variablenfreie Formeln erster Stufe von Veriablenbelegungen unabhängig sind.

2. Mit der (noch unbewiesenen) *Vollständigkeit der AL-Klauseltableaus* folgt, daß es ein geschlossenes AL-Klauseltableaus T' für S' gibt.

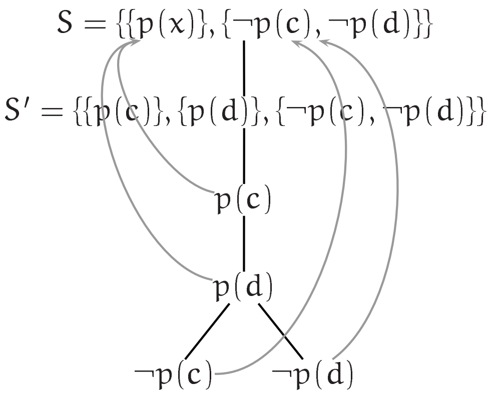
3. Für jedes in T' verwendete $C^{i\prime}$ sei τ^i so, daß $C^i\tau^i = C^{i\prime}$ für ein $C^i \in S$. Damit definiere $\tau = \bigcup_{C^{i\prime} \text{ verwendet in } T'} \tau^i$.
   T'' ist wie T' aber ohne Substitutionen. Da $T''\tau = T'$, ist T'' geschlossenes Klauseltableau.

   Der Satz über Lifting besagt, daß die Klauseltableauprozedur ein geschlossenes T für S findet.

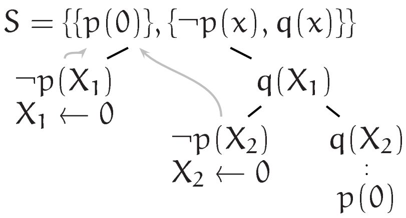
∎

Es fehlt noch Schritt 2., die Vollständigkeit der AL-Klauseltableaus. Diese wird aus einem allgemeineren Resultat folgen, das im folgenden Kapitel bewiesen wird.

Nun noch ein Beispiel zu den Mengen S und S' im obigen Beweis. S' geht über $\Sigma' = \{p(c), p(d)\} \subset At_\Sigma$ mit $\Sigma = \langle\{p/1\}, \{c/0, d/0\}\rangle$. S' unerfüllbar über Σ' und das AL-Tableau für S' ist geschlossen. Die gestrichelten Pfeile deuten das zugehörgige Tableau für S an.



Ist eine Klauselmenge S erfüllbar, dann gibt es auch ein Herbrandmodell von S. Prinzipiell kann man (Herbrand-)Modelle an den Substitutionen in unendlichen, fairen und offenen Zweigen ablesen, zum Beispiel an dem ganz rechten Zweig im folgenden Tableau:



Offenbar ist $\langle\{0\}, I\rangle$ mit $I(p) = I(q) = \{0\}$ ein Herbrand-Modell von S. Im allgemeinen können Modelle jedoch ein unendliches Universum besitzen. Wir sprechen dann von *unendlichen Modellen*. Man kann Klauselmengen konstruieren, die nur unendliche Modelle haben (Übung; Hinweis: formalisiere zum Beispiel eine lineare, irreflexive Ordnung $<$, die kein maximales Element hat).

$S' = \{\{p(0)\}, \{\neg p(x), p(s(x))\}\}$ hat keine endlichen Herbrand-Modelle, aber $\langle\{s^n(0) | n \in \mathbb{N}\}, I\rangle$ mit $I(p) = \{s^n(0) | n \in \mathbb{N}\}$ ist offenbar ein Herbrand-Modell von S'.

Die Erkennung bestimmter Teilklassen erfüllbarer Formeln und die Generierung von Modellen aus gescheiterten Tableaubeweisen ist ein aktuelles Forschungsgebiet; siehe dazu Caferra and Zabel [1993].

Kapitel 5

Verfeinerungen von Klauseltableaus

Die uniforme Gestalt universeller Klauseln ermöglicht die einfache Definition von Verfeinerungen von Klauseltableaus bzw. der Klauseltableauprozedur.

Grundidee: Schränke die Menge aller Klauseltableaus so ein, daß es immer noch für jede unerfüllbare Klauselmenge ein geschlossenes Tableau gibt.

Die Vollständigkeit bleibt gewahrt, aber die Anzahl der zu betrachtenden Tableaus nimmt ab, d.h. der Suchraum der Tableauprozedur wird kleiner.

Konkret bedeutet dies die Eliminierung von Choicepoints in:

select_clause(S)
select_pair(B)

Und die Konkretisierung von:

select_branch(T)

Natürlich muß man die Korrektheit von Verfeinerungen nicht mehr zeigen, da diese aus der Korrektheit der Tableauprozedur folgt. Es genügt außerdem, die Grundvollständigkeit von Verfeinerungen zu zeigen, sofern diese rein aussagenlogisch definiert sind (dies folgt aus der Lifting-Eigenschaft von Klauseltableaus).

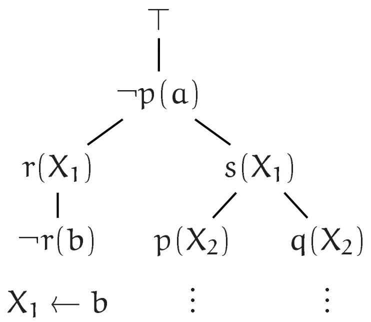
5.1 Konnektionstableaus

Die im folgenden definierte, sogenannte Konnektionsverfeinerung geht auf Andrews (1981) und Bibel (1979) zurück und liegt auch der Arbeit Letz *et al.* [1992] zugrunde.

Definition 5.1 *Ein* Konnektionstableau *ist ein Klauseltableau, worin jeder innere mit* L *markierte Knoten (außer* $\top$*) ein mit* L' *markiertes Blatt als unmittelbaren Nachfolger hat, so daß* $\langle L, L' \rangle$ *komplementäres Paar ist.* ■

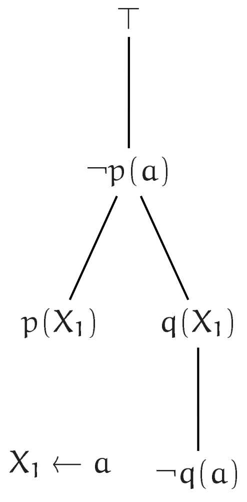
Beispiel 5.2

$$S = \{\{p(x), q(x)\}, \{r(x), s(x)\}, \ \{\neg p(a)\}, \{\neg q(a)\}, \{\neg r(b)\}, \{\neg s(b)\}\}$$



Dies ineffiziente Tableau wird durch die Konnektionsverfeinerung verhindert. Es wird stattdessen z.B. das folgende Tableau erzeugt.

$$S = \{\{p(x), q(x)\}, \{r(x), s(x)\}, \ \{\neg p(a)\}, \{\neg q(a)\}, \{\neg r(b)\}, \{\neg s(b)\}\}$$



ϵ

Nach Auswahl der initialen Klausel ist die Auswahl der übrigen Klauseln deterministisch. ■

Konnektionstableaus verkörpern die Idee, daß der jeweils folgende Beweisschritt etwas mit dem vorangegangenen zu tun haben muß. Dies ergibt nicht immer den kürzestmöglichen Beweis, schränkt aber den Suchraum deutlich ein.

Konnektionstableaus kann man alternativ als Verschärfung der Link-Optimierung ansehen.

Ihre Implementierung erfordert Modifikation von select_clause, da Erweiterung und Substitution gekoppelt sind; select_clause enthält nun einen Teil der bisher in select_pair enthaltenen Funktionalität. Wir gehen von einer Implementierung durch Backtracking aus (inkrementelle beschränkte Tiefensuche).

Prozedurale Beschreibung:

```
function select_clause(B,S:branch):
  ⟨clause,substitution⟩;
var S': branch;
begin
  if B = [] then
    return(or(S×{ε}))
  else begin
    S' := {⟨C,σ⟩ |C∈S,⟨C,last(B)⟩ Link mit σ};
    return(or(S'))
  end
end;

function extend(B,S:branch):
  ⟨tableau,substitution⟩;
    ...
  return(⟨U,σ⟩)
end;

/* connection tableaux */

T := first( extend([],S) );
  ...
    else
      begin
      ⟨U,σ⟩ := extend(B,S) ;
      T := apply_subst(σ,append(T,U))
      end
  ...
```

Wir geben an, was gegenüber der Klauseltableauprozedur geändert werden muß. Hauptsächlich muß man dafür sorgen, daß bei der Erweiterung die

Konnektionsbedingung erfüllt wird. Da `extend` neue Literale an B anhängt, muß C mit `last(B)` ein Link bilden. Die notwendige Substitution wird von `select_clause` mit zurückgegeben. Nur für Klauseln, die die Konnektionsbedingung erfüllen, wird ein Choicepoint angelegt. Bei der initialen Erweiterung braucht die Konnektionsbedingung nicht erfüllt werden, was im Fall `B = []` ausgedrückt ist. Die Funktion `extend` muß nun auch die benötigte Substitution zurückgeben, die in der Hauptschleife im Erweiterungsfall auf das Tableau angewandt wird.

Die übrigen Auswahlfunktionen sind direkte Vereinfachungen der früheren Nichtklauselversionen für Backtracking. Weiterhin müssen geeignete *fails* für einen Vollständigkeitsmodus programmiert werden.

Jeder Erweiterungsschritt ermöglichst direkt einen Abschluß und zwar mit einem Blatt des erweiterten Tableaus.[1] (Bei unserer Implementierung werden auch diese Abschlüsse explizit mit select˙mode = close ausgeführt (und nicht implizit durch den Erweiterungsschritt). Andere Abschlüsse, also solche, die nicht auf den Link benutzen auf dem der Erweiterungsschritt berueht, nennen wir *Reduktionsschritt*.

Beispiel 5.3 Bei der Konstruktion eines Tableaus für die Klauseln $\{\neg q\}$, $\{q, r\}$ und $\{\neg r, q\}$ werde zunächst $\{\neg q\}$ auf dem Tableau plaziert. Dann wird das Tableau mit $\{q, r\}$ erweitert und Anschließend mit $\{\neg r, q\}$. Der Abschluß des nun noch offenen Astes, der $q, r, \neg q$ enthält, ist ein Reduktionsschritt. ■

Satz 5.4 *Konnektionstableaus sind* nicht *beweiskonfluent.*

Beweis: Betrachte $S = \{\{p\}, \{\neg p\}, \{q\}\}$ und $\{q\}$ sei als erste Klausel selektiert. Man kann ohne weiteres auch Beispiele finden, die unabhängig von der Startklausel sind Letz [1993]. ■

Dies bedeutet, daß Konnektionstableaus mit deterministischer Implementierung von `select_clause` unvollständig sind, sogar wenn letzteres fair implementiert ist. Insbesondere ergeben offene, nicht erweiterbare Zweige kein Modell.

5.2 Schwache Konnektionstableaus

Hier gilt immer noch die Einschränkung, daß Erweiterungsschritte nur mit Klauseln erlaubt sind, die einen Link in den aktuellen Ast haben. Es sind jedoch Links zu einem *beliebigem* Ast-Literal zulässig (nicht notwendigerweise mit dem Blatt).

[1] Dem ist nicht so bei Link-Optimierung im Nichtklauselfall.

Definition 5.5 *Ein* schwaches Konnektionstableau *ist ein Klauseltableau, in dem es zu jedem mit* L *markierten Knoten – außer der Wurzel und den direkten Nachfolgern der Wurzel – ein Literal* L' *auf dem Pfad zwischen* L *und der Wurzel gibt, so daß* $\langle L, L' \rangle$ *ein komplementäres Paar ist.* ■

Definition 5.6 *Eine Klauselmenge ist* **minimal unerfüllbar** *(mU), falls sie unerfüllbar ist und jede ihrer echten Teilmengen erfüllbar.* ■

Schwache Konnektionstableaus sind beweiskonfluent, falls die Start-Klausel in einer minimal unerfüllbaren Teilmenge von S vorkommt.

5.3 Regularität

Definition 5.7 *Ein* reguläres Tableau *ist ein Klauseltableau, in dem auf keinem Zweig dasselbe Literal mehr als einmal vorkommt.* ■

Beispiel 5.8 Reguläre Tableaus können sinnlose identische Instantiierungen verhindern, wie man in Abbildung 5.1 sieht. Die erste mögliche Substitution für den Abschluß des mittleren Zweigs würde den rechten Zweig irregulär machen. ■

Die Implementierung von Regularität in der Prädikatenlogik ist leider nicht ganz einfach. Das Problem ist, daß spätere, noch unbekannte Substitutionen die Regularität des aktuellen Zweigs verletzen können.

Seien auf einem aktuellen geschlossenen Zweig sowohl $p(X)$ als auch $p(Y)$ vorhanden. Eine spätere Instantiierung könnte X mit Y unifizieren. Das kann aber jetzt noch nicht entschieden werden.

Eine mögliche Implementierung wurde von Letz *et al.* [1992] vorgeschlagen: Für jedes unifizierbare Paar von Literalen $P(t_1, \dots, t_m)$, $P(t'_1, \dots, t'_m)$ auf einem Zweig wird eine *Disjunktion von Ungleichungen* $t_1 \neq t'_1 \vee \dots \vee t_m \neq t'_m$ erzeugt, die von allen zukünftigen Substitutionen beachtet werden muß, das heißt, keine Substitution darf bewirken, daß eine Disjunktion von Ungleichungen der Form $t_1 \neq t_1 \vee \dots \vee t_m \neq t_m$ entsteht.

Es gibt dann eine weitere Stelle (außer den Stellen, wo das Enthaltensein in $m(i)$ überprüft wird), an der in der Tableauprozedur ein `fail` erzeugt wird, nämlich sobald eine Formel der Gestalt $t_1 \neq t_1 \vee \dots \vee t_m \neq t_m$ auf einem Zweig auftaucht.

Diese Lösung kommt ohne die Speicherung aller schon geschlossenen Zweige aus. In einem gewissen Sinne werden gerade die für frühere Abschlüsse benötigten Substitutionen gespeichert.

In obigem Beispiel kann man nach der Erweiterung mit $\{\neg p(X_2), p(s(X_2))\}$ die Ungleichung $s(X_2) \neq s(0)$ zur Wurzel des Tableaus hinzunehmen. Dies verhindert die Substitution $\{X_2 \leftarrow 0\}$.

$$S = \{\{p(0)\}, \{\neg p(x), p(s(x))\}, \{\neg p(s(s(0)))\}\}$$

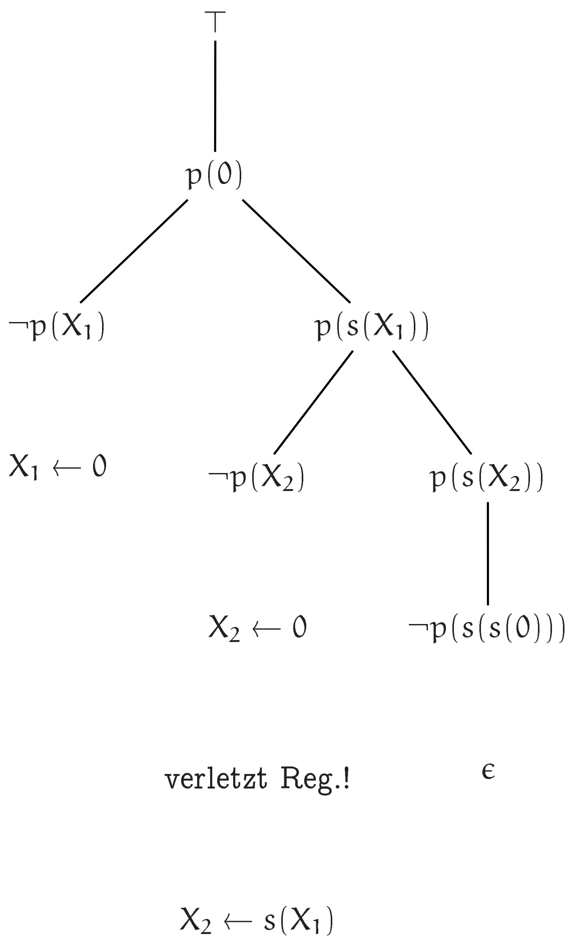


Abb. 5.1: Vorteil durch die Regularitätseinschränkung.

Satz 5.9 *Sei* S *eine unerfüllbare PL-Klauselmenge über* Σ. *Dann findet die Konnektionstableauprozedur ein geschlossenes, reguläres Konnektionstableau* T *für* S.

Lifting geht wie früher, also genügt die AL-Vollständigkeit von regulären Konnektionstableaus. Zunächst braucht man ein Lemma:

Lemma 5.10 *[Loveland, 1978, Lemma 2.3.2, S. 63]*
Sei S *eine minimal unerfüllbare AL-Klauselmenge. Ferner sei*

(a) $C \in S$ *eine Klausel in* S,

(b) $D \subseteq C$ *eine Teilklausel von* C,

(c) S'' *eine Teilmenge von* $S' = (S - \{C\}) \cup \{D\}$.

Ist S'' *mU, dann gilt:*

(1) $D \in S''$,

(2) $D \not\subseteq D''$ *für alle* $D \neq D'' \in S''$.

Beweis:

1. Angenommen, D ist nicht S. Dann gilt $S'' \subseteq S' - \{D\} = S - \{C\} \subsetneq S$. Also ist S nicht mU. Widerspruch.

2. Angenommen $D \subseteq D'' \in S''$, $D \neq D''$. Da S'' mU, hat $S'' - \{D''\}$ ein Modell. Dieses ist auch Modell von D, damit aber auch von D'' und also von S''. Widerspruch.

■

Beweis: *Vollständigkeit AL-Konnektionstableaus mit Regularität.*
S ist o.B.d.A. minimal unerfüllbar.

Wir zeigen per Induktion über die Anzahl n der Literal*auftauchen* in S:

> Für jede Nicht-Einerklausel $C \in S$ gibt es ein geschlossenes reguläres Konnektionstableau für S, in dem der erste Erweiterungsschritt mit C erfolgt.

Dies genügt, da Fälle, in denen es keine Nicht-Einerklauseln gibt, trivial sind (s.u.).

$\mathbf{n = 1}$: Tritt nicht auf, da S mU.

$\mathbf{n = 2}$: Wegen mU ist S von der Form $\{\{p\}, \{\neg p\}\}$. Trivial.

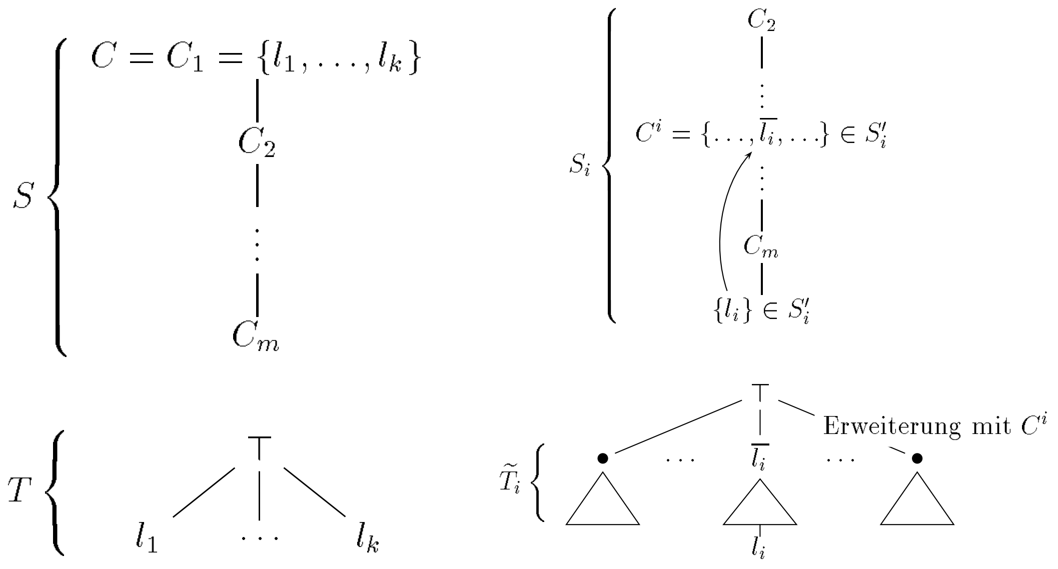


Abb. 5.2: Illustration zum Beweis von Satz 5.9.

n > 2: Sei $S = \{C_1, \ldots, C_m\}$. Wegen mU muß es eine Nicht-Einerklausel geben. O.B.d.A. sei $C = C_1 = \{l_1, \ldots, l_k\} \in S$ eine solche.

Sei T das Tableau, das nach dem ersten Erweiterungsschritt mit $C = C_1$ entstanden ist. T is offensichtlich regulär und konnektiert (siehe Abbildung 5.2 links).

Sei $S_i = \{\{l_i\}, C_2, \ldots, C_m\}$ für $1 \leq i \leq k$. Aus dem Lemma folgt: $\{l_i\}$ kommt in einer mU Teilmenge S_i' von S_i vor. Wegen mU von S_i' hat $\{l_i\}$ Link mit $C^i \in S_i'$.

Ist C^i Einerklausel, wird der i-te Ast von T mit $C^i \in S$ erweitert und geschlossen.

Andernfalls hat S_i' weniger Literalvorkommen als S und die Induktionshypothese liefert ein reguläres Konnektionstableau T_i für $S_i \supseteq S_i'$, in dem zuerst mit C^i erweitert wird.

Der Wald $\widetilde{T}_i$ entstehe aus den T_i durch Löschung der Wurzeln $\top$ (siehe Abbildung 5.2 rechts). Erweitere jeden noch offenen Ast in T mit Blatt l_i durch „Ankleben" von $\widetilde{T}_i$.

Das neue Tableau T' ist geschlossen, da jedes der $\widetilde{T}_i$ geschlossen ist. T' ist ein Konnektionstableau, da jedes $\widetilde{T}_i$ eines ist und durch Wahl der Start-Klauseln C^i. Jedes der $\widetilde{T}_i$ ist regulär.

Aus dem Lemma folgt, daß l_i nur in der Einerklausel in S_i' vorkommt, und also höchstens als Blatt in $\widetilde{T}_i$.

Ersetze alle Erweiterungsschritte mit $\{l_i\}$ in $\widetilde{T}_i$ durch einen Abschluß mit dem obenstehenden l_i in T.

Ergebnis: ein reguläres Konnektionstableau. ∎

5.4 Geordnete Tableaus und Tableaus mit Selektionsfunktion

Die Ziele von Ordnungen und Selektionsfuntkionen sind:

- Umgehung der Problematik, daß Startklausel in mU Menge sein muß (welche i.a. unbekannt ist).
- Stärkere Einschränkung der Auswahlregel als bei schwachen Konntektionstableaus. (Ist S mU bedeutet die schwache Konnektionsbedingung evtl. wenig Einschränkung.)
- Beibehaltung der Beweiskonfluenz.

Die Lösungsansätze sind:

- Verschärfe die Linkbedingung schwacher Konnektionstableaus.
- Erlaube unter gewissen Umständen eine Erweiterung mit Klauseln ohne Link.

Definition 5.11 *Eine* $\mathcal{A}$-Ordnung *ist eine irreflexive, transitive Relation* $< \subseteq At_\Sigma \times At_\Sigma$ *mit:*

Für alle $p, q \in At_\Sigma$ *und Substitutionen* σ*: aus* $p < q$ *folgt* $p\sigma < q\sigma$.
(Substitutivität)

∎

Definition 5.12 *Seien* L, L' *Literale,* C *Klausel. Dann gilt* $L < L'$ *gdw* $atom(L) < atom(L')$.
L *ist* $<$-***maximal*** *in* C *gdw es kein* $L' \in C$ *gibt mit* $L' > L$. ∎

Anstatt der Erweiterung von $<$ von Atomen auf Literale, kann $<$ direkt auf Literalen definiert werden, was eine leichte Verstärkung bringt. Alles folgende geht dann immer noch und man spricht von $\mathcal{L}$-**Ordnungen**.

Aus der Definition folgt sofort, daß $L \nless L'$ und $L' \nless L$, falls $\{L, L'\}$ unifizierbar ist. Denn ist $\{L, L'\}$ unifizierbar mit mgu σ, dann folgt mit der Substitutivität z.B. aus $L < L'$, daß $L\sigma < L'\sigma$. Das ist unmöglich, da $L\sigma = L'\sigma$ und $<$ irreflexiv ist. $\mathcal{A}$-Ordnungen sind also stets nur partielle Ordnungen.

Definition 5.13 *Die* Termtiefe $\tau(t)$ *für* $t \in Term_\Sigma$ *ist die maximale Differenz zwischen sich öffnenden und schließenden Klammern in einem zusammenhängendem Teilwort von* t *plus 1.*

Entsprechend ist die Tiefe $\tau_{max}(s,t)$ *eines Unterterms* s *in einem Term* t *definiert, aber beschränkt auf Teilwörter, die direkt links von* s *enden.* ■

Definition 5.14 *Für* $p, q \in At_\Sigma$ *ist* $p \prec q$ *gdw* $\tau(p) < \tau(q)$ *und für alle* $x \in Var(p)$ *ist* $x \in Var(q)$ *und* $\tau_{max}(x,p) < \tau_{max}(x,q)$. ■

Übung: Man zeige, daß $\prec$ eine A-Ordnung ist.

Beispiel 5.15

- $p(x,x) \prec q(f(x),y)$
- $p(x,f(a)) \not\prec q(x,f(x))$ (da gleiche Termtiefe)
- $p(x,y) \prec r(g(x,y))$
- $p(x,a) \not\prec p(f(a),x)$ (gleiche Tiefe von x in den beiden Termen; dies führt zum Beispiel bei Anwendung der Substitution $\{x/f(a)\}$ zu einem Problem)

■

Zum Teil können sinnvolle Ordnungen aus konkreten Problemen extrahiert werden: In der Software-Verifikation werden meist die abstrakten Datentypen der Objektprogramme mittels Gleichungen oder Äquivalenzen definiert, z.B.:

append(cons(a,x),y) = cons(a,append(x,y))
x ≠ nil → cons(car(x),append(cdr(x),y)) = append(x,y)

Aus derartigen Regeln kann zunächst eine Ordnung der Funktions- bzw. Prädikatssymbole generiert werden. Ist f und g das oberste Funktions- bzw. Prädikatssymbole links bzw. rechts des Gleichheitszeichens. Dann definiere $f > g$.

Regeln der Form $\phi \rightarrow \sigma = \tau$, falls $\tau\nu$ mit Unterterm von σ unifizierbar ist, werden nicht beachtet, damit keine unintuitiven Ergebnisse entstehen (wie bei der zweiten Regel oben).

Dann wird $<$ auf $Term_\Sigma$, At_Σ erweitert mittels:
$s = f(t_1, \ldots, t_n) < g(s_1, \ldots, s_m) = t$ gdw

1. $f < g$, oder $f = g$, $n = m$ und $t_i < s_i$ für $i = 1, \ldots, n$;
2. $Var(s) \subseteq Var(t)$.

In schwachen Konnektionstableaus können nun Links auf $<$-maximale Literale in der erweiternden Klausel eingeschränkt werden. Das ist aber unvollständig, wie das folgende Beispiel zeigt (maximale Literale gekennzeichnet):

$S = \{\{A, \boxed{B}\}, \{\neg A, \boxed{B}\}, \{\boxed{\neg B}\}\}$, $A < B$ und Start mit einer der ersten beiden Klauseln. In keinem Fall läßt sich das Tableau mit einem geordneten Erweiterunfsschritt schliessen.

S ist aber sogar minimal unerfüllbar. Es gibt auch Beispiele, die unabhängig von der Startklausel sind Hähnle and Klingenbeck [1996].

Wie schon angedeutet, muß man auch bestimmte Erweiterungsschritte ohne Link mit dem zu erweiternden Zweig zulassen. Dies hat zunächst nichts mit Ordnungen zu tun, sonder gibt eine Alternative Formulierung von schwachen Konnektionstableau, die den Nachteil vermeidet, daß die Startklausel aus einer mU Teilmenge stammen muß (man vergleiche mit Definition 4.1).

Definition 5.16 *Ein* schwaches Konnektionstableau mit Neustart *für eine endliche Klauselmenge* $S = \{C_1, \ldots, C_m\}$ *ist ein endlichverzweigender, mit Literalen markierter Baum, der konstruiert ist mittels:*

1. Initialer Schritt: *Der Baum mit genau einem Knoten markiert mit* $\top$ *ist ein Klauseltableau für* S.

2. Erweiterungsschritt: T *sei ein Klauseltableau für* S, B *ein Ast von* T, $C = \{L_1, \ldots, L_n\}$ *eine Klausel in* S *und* σ *eine Substitution, so daß es*

   (i) *ein* $L \in B$ *oder*

   (ii) *ein* $L \in C' \in S$

   gibt und $\langle L\sigma, L_i\sigma\rangle$ *für ein* $1 \leq i \leq n$ *ein komplementäres Paar ist.*

   Ein Klauseltableau für S *entsteht aus* $T\sigma$, *indem* B *durch* n *neue Nachfolger mit Markierungen* $L_1\sigma, \ldots, L_n\sigma$ *erweitert wird.*

3. Reduktionsschritt: T *sei ein Klauseltableau für* S, B *ein Ast von* T, L *das Blatt von* T *und* L' *ein anderes Literal auf* B, *und* σ *sei eine Subsitution.*

   Ist $\langle L\sigma, L'\sigma\rangle$ *ein komplementäres Paar, dann ist* $T\sigma$ *ein Klauseltableau für* S.

Die Klausel C *in 2. (b) heißt* Neustartklausel. ∎

Die Einschränkungen 2. (a) und (b) gelten insbesondere auch für den ersten Erweiterungsschritt (dieser ist also in gewisser Weise der erste Neustart).

Schwache Konnektionstableaus mit Neustart sind die Basis für geordnete Tableaus:

Definition 5.17 *Ein* geordnetes Tableau *für eine endliche Klauselmenge* $S = \{C_1, \ldots, C_m\}$ *ist ein schwaches Konnektionstableau mit Neustart, wobei zusätzlich gilt:*

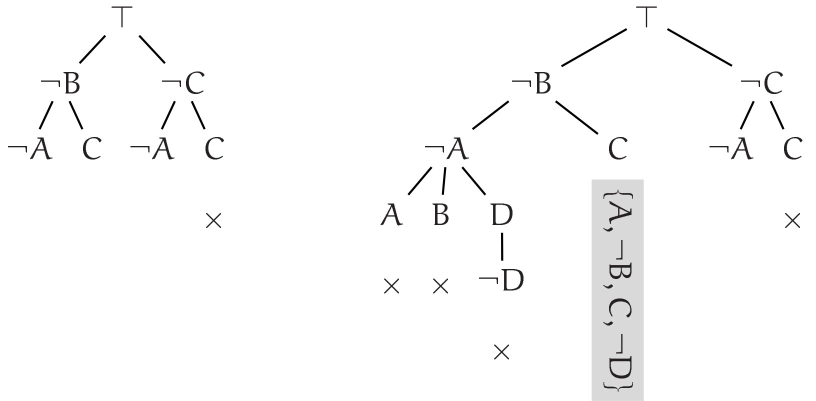


Abb. 5.3: Das in Beispiel 5.18 beschriebene Tableau.

1. *In 2.(a) und 2.(b) ist* $L_i\sigma$ *<-maximal in* $C\sigma$ *und*

2. *in 2.(b) ist* $L\sigma$ *<-maximal in* $C'\sigma$.

■

Beispiel 5.18 Sei $S = \{\{\neg B, \boxed{\neg C}\}, \{\boxed{A}, B, D\}, \{\neg A, \boxed{C}\}, \{\boxed{\neg D}\}\}$ mit Ordnung $D < B < A < C$.

Nur die beiden Klauseln, in denen C maximal ist sind Neustartklauseln, kommen also für den ersten Schritt in Frage.

Da B nirgends maximal auftaucht, bleibt dies auch beim zweiten Schritt so. Das in Abbildung 5.3 links dargesllte Tableau ist also bis auf die Reihenfolge der beiden Klauseln determiniert. Der rechte Zweig ist schon geschlossen.

Im linken Zweig ist nur die Klausel der Länge 3 anwendbar (Regularität), dann die Einerklausel.

Auf den linken der beiden noch offenen Zweige ist nun keine geordnete Extensionsregel mehr anwendbar!

In der Tat läßt sich dieser wie in Abbildung 5.3 rechts angedeutet zu einem Modell erweitern. Dazu ist keine weitere Regelanwendung notwendig. ■

Satz 5.19 (Hähnle and Klingenbeck [1996]) *Sei < eine beliebige A-Ordnung; und sei S eine unerfüllbare Menge von PL-Klauseln. Dann gibt es geschlossenes reguläres <-geordnetes Klauseltableau für S.*

Außerdem gilt: geordnete, reguläre Klauseltableaus sind beweiskonfluent.

Beweis: Lifting geht wegen (Substitutivität) genau wie früher, also folgt der Satz sofort aus der Vollständigkeit für die AL. Sei S nun AL-Klauselmenge.

Ein Zweig B heißt *saturiert*, falls er offen ist und für alle $C \in S$:

i. C wurde auf B verwendet oder

ii. C verletzt die Bedingungen (a) und (b) aus Tableaukonstruktionsregel 2 oder

iii. C verletzt die Regularität.

Gibt es für S kein geschlossenes geordnetes Klauseltableau, dann gibt es offenbar eines mit saturiertem Zweig B.

I sei die durch Literale auf B definierte partielle Interpretation von S:

$$I(p) = \begin{cases} W & \text{falls } p \text{ auf } B \\ F & \text{falls } \neg p \text{ auf } B \end{cases}$$

I ist wohldefiniert, da B offen ist.

Die Klauselmenge $S' \subseteq S$ sei definiert als $\{C \in S \mid I \not\models C\}$.

Auf Klauseln in S' trifft nach Definition von S' und Regularität genau (ii) zu.

Sei I' Menge der $<$-maximalen Literale in S'.

Wegen Tableaukonstruktionsregel 2.(b) gilt: I' ist wohldefiniert, sonst wäre Saturiertheit von B verletzt. Genauso, wegen 2.(a), ist $I \cup I'$ wohldefiniert.

Also $I' \models S'$ und damit $I \cup I' \models S$. Widerspruch!

Die Beweiskonfluenz folgt unmittelbar aus der Tatsache, daß aus jedem saturierten Zweig ein Modell konstruiert werden kann. ■

Beispiel 5.20 Im vorogen Beispiel (linker offener Zweig) ist in der Notation des Beweises: $I = \{\neg B, C\}$ und $S' = \{\{A, B, D\}, \{\neg D\}\}$. Damit ergibt sich $I' = \{A, \neg D\}$. ■

Wie aus dem Beweis ersichtlich, muß S nicht endlich sein.

Tableaus mit Selektionsfunktion Jede A-Ordnung $<$ induziert **Selektionsfunktion** $f_<$ auf Klauseln:

$$f_<(C) = \{L \in C \mid L \ < \text{-maximal in } C\}$$

Alle Resultate gelten immer noch falls

- „L $<$-maximal in C“ ersetzt wird durch „$L \in f(C)$“, wobei
- f jeder Klausel eine beliebige nichtleere Teilmenge ihrer Literale zuweist.

Dies stellt eine echte Verallgemeinerung der geordneten Tableaus dar, denn z.B. wird $f(\{A, B\}) = A$, $f(\{B, C\}) = B$, $f(\{A, C\}) = C$ nur von einer zyklischer Ordnung induziert. Also kann nicht jede Selektionsfunktion als $f_<$ für eine geeignete A-Ordnung dargestellt werden.

Die Implementierung geordneter Tableaus bzw. Tableaus mit Selektionsfunktion hat die gleiche Problematik und Lösung wie die Regularität, siehe Hähnle and Pape [1997].

5.5 Probleme von Verfeinerungen

Tableauverfeinerungen können den Suchraum erheblich einschränken. Leider gibt es auch einige Nachteile:

1. Schon im AL-Fall sind Konnektionstableau ja nicht mehr beweiskonfluent. Die Folge ist, daß in vollständigen Implementierungen von Konnektionstableaus die Funktion `select_clause` nicht deterministisch gemacht werden darf. Bei Implementierung mit inkrementeller Tiefensuche muß über alle Klauseln aus S, die die Konnektionsbedingung erfüllen, Backtracking initiiert werden. Dies ist ein fundamentaler Unterschied zu beweiskonfluenten Tableauvarianten.

   Trotz zusätzlicher Choicepoints kann die Verwendung der Konnektionseinschränkung Vorteile bei der Beweissuche bringen, weil durch sie die überhaupt für die Erweiterung in frage kommenden Klauseln unter Umständen stark eingeschränkt werden.

   Beweise, die mit der Konnektionseinschränkung gefunden werden, beinhalten typischerweise mehr Backtracking, sind oft aber weniger redundant als Beweise, die ohne diese Einschränkung gefunden werden.

2. Ebenfalls aus der fehlenden Beweiskonfluenz ergibt sich das Problem, daß ein offener Zweig, auf den keine Regel mehr anwendbar ist, unter der Konnektionsverfeinerung i.a. kein Gegenbeispiel darstellt. Bei Klauseltableaus (und Nichtklauseltableaus) kann man in der AL aus solchen Zweigen wie im Beweis von Satz 5.19 sofort ein Modell für die initiale Formel ablesen.

3. Weiterhin kann man folgern, daß im AL-Fall nicht beweiskonfluente Kalküle nicht sinnvoll einzusetzen sind, da es keine Substitutionen gibt und somit alle Choicepoints auch ohne inkrementelle Tiefensuche bei Klauseltableaus eliminierbar sind. Konnektionstableaus führen somit unnötigerweise Choicepoints ein und entfernen keine dafür.

4. Es gibt eine Formelklasse Φ_n, deren kürzester Beweis mit Konnektionstableaus exponentiell länger ist als ihr kürzester Beweis mit Klauseltableaus (entsprechendes gilt auch für geordnete Tableaus):

   $C_n = \{\{L_1, \ldots, L_n\} \mid L_i \in \{p_i, \neg p_i\}, 1 \leq i \leq n\}$

   $\Phi_n = C_n \cup \{\{p_i, \neg p_i\} \mid 1 \leq i \leq n\}$

   Beispiel für $n = 3$:

   $$\begin{array}{llll} \Phi_3 = \{ & \{p, q, r\}, & \{\neg p, q, r\}, & \{p, \neg p\}, \\ & \{p, q, \neg r\}, & \{\neg p, q, \neg r\}, & \{q, \neg q\} \\ & \{p, \neg q, r\}, & \{\neg p, \neg q, r\}, & \{r, \neg r\} \end{array}$$

$\{p, \neg q, \neg r\}, \{\neg p, \neg q, \neg r\} \qquad \}$

Der kürzeste Klauseltableaubeweis T_n für Φ_n (siehe Abbildung 5.4) hat $n2^n + 2^{n+1} - 1$ Knoten, und $|\Phi_n| = n2^n + 2n$ (Anzahl der Literale), also ist $|T_n|$ linear in $|\Phi_n|$.

In den ersten n Ebenen werden die tautologischen Klauseln in Φ_n expandiert, so daß in jedem Zweig die Literale genau einer Klausel in C_n vorkommen.

Offenbar verletzt dieser Beweis die Konnektionsbedingung. Um diese einzuhalten, muß schon als zweite Regelanwendung eine Klausel aus C_n gewählt werden. Die Verwendung weiterer tautologischer Klauseln verkürzen den Beweis nicht mehr, weil ein Literal jeder tautologischen Klausel schon in ψ und damit auf jedem Zweig ist. Ein Konnektionstableaubeweis von Φ_n ist mithin also mindestens so lang wie ein Klauseltableaubeweis von C_n. In D'Agostino [1992] wurde gezeigt, daß ein solcher mindestens $(n-1)!$ Knoten hat.

Der kürzeste Konnektionstableaubeweis für Φ_n hat somit auch mindestens $(n-1)!$ Knoten, ist also nicht polynomiell in $|\Phi_n|$.

5. Die Kombination von für sich genommen vollständigen Verfeinerungen kann unvollständig sein wie die naive Kombination von schwachen Konnektionstableaus mit Ordnungen zeigt.

5.6 Weitere Verfeinerungen

Konnektionstableaus spezifizieren keinerlei Einschränkung für die Zweigauswahlregel `select_branch`. Verschiedene Festlegungen dieser ergeben weitere bekannte Kalküle in Tableauformulierung.

5.6.1 Die Konnektionsmethode

Wir präsentieren die AL-Variante der Konnektionsmethode (Bibel [1981, 1987]) für Formeln in KNF. Die allgemeine Form der Konnektionsmethode ist für beliebige Formeln erster Stufe definiert. Eine Formel in KNF der Form

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} L_{i,j}$$

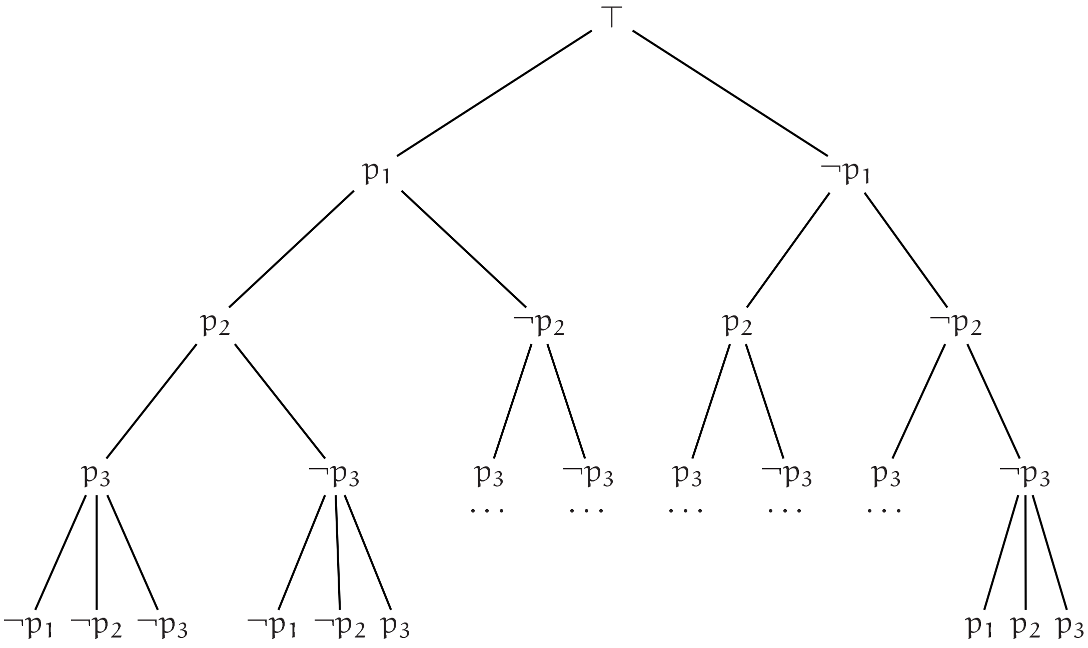


Abb. 5.4: Klauseltableaubeweis für Φ_3.

wird in *Matrixschreibweise* notiert:

$$\begin{bmatrix} L_{1,1} & L_{2,1} & \cdots & L_{n,1} \\ \vdots & \vdots & & \vdots \\ L_{1,m_1} & L_{2,m_2} & \cdots & L_{n,m_n} \end{bmatrix}$$

Ein *Pfad* durch eine Matrix M ist eine Menge von Literalen der Gestalt

$$\{L_{1,i_1}, \ldots, L_{n,i_n}\}$$

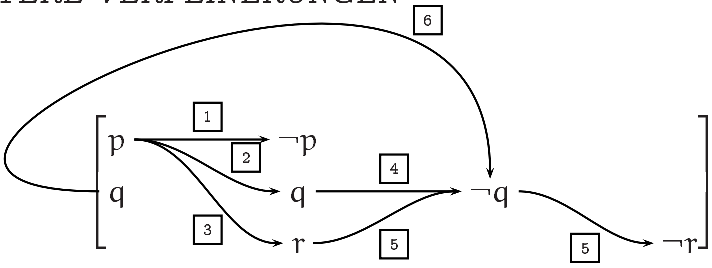
wobei $1 \leq i_j \leq m_j$ für alle $1 \leq j \leq n$.

Ein *Pfad* heißt *komplementär*, falls er ein komplementäres Paar $\{p, \neg p\}$ als Teilpfad enthält. Eine *Matrix* heißt *komplementär*, falls alle Pfade durch sie komplementär sind. Insbesondere ist jede Matrix für eine Klauselmenge, die $\{\}$ enthält, komplementär.

Satz 5.21 (Bibel) *Die Matrix einer AL-Klauselmenge S ist komplementär gdw S unerfüllbar ist.*

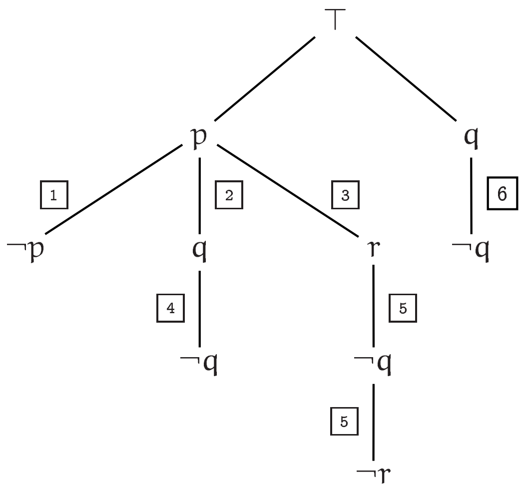
Die Konnektionsmethode besteht in einer systematischen Prozedur für die Feststellung der Komplementarität einer Matrix.

Beispiel 5.22 $S = \{\{p, q\}, \{\neg p, q, r\}, \{\neg q\}, \{\neg r\}\}$.



Die Pfade werden in der in den kleinen Zahlen angedeuteten Reihenfolge durchsucht. Die ersten beiden Teilpfade sind $\{p\}$ und $\{q\}$, der erste wird dann zu $\{p, \neg p\}$ erweitert ($\boxed{1}$); da alle Pfade, die diesen enthalten, auch widersprüchlich sind, braucht von dort nicht weitergesucht zu werden, und als nächstes wird $\{p\}$ zu $\{p, q\}$ erweitert ($\boxed{2}$).

Die (Teil-)pfade in obiger Matrix entsprechen den (Teil-)Zweigen in folgendem Klauseltableau für S:



■

Jede Suchstrategie für komplementäre Pfade entspricht also einer Auswahlstrategie für Klauseltableaus.

Die Kalkülfamilie „Konnektionsmethode“ kann damit charakterisiert werden als:

Konnektionstableau + beliebige Depth-first-Zweigauswahlregel

Die Konnektionsmethode ist auch für Formeln erster Stufe definiert. In diesem Fall muß die Implementierung mit inkrementeller Tiefensuche erfolgen. Der Vollständigkeitsmodus ist dann zum Beispiel die maximale Anzahl von Kopien einer Spalte mit frischen Variablen. Details entnehme man Bibel [1987].

Es gibt auch eine Variante der Konnektionsmethode für beliebige Formeln in NNF. Matrizen werden dahingehend verallgemeinert, daß ihre Elemente selbst wieder Matrizen sein dürfen. Zum Beispiel kann die Formel

$$\neg p \wedge (p \vee (\neg q \wedge (p \vee q)))$$

durch die Matrix

$$\begin{bmatrix} & p \\ \neg p & \begin{bmatrix} & p \\ \neg q & q \end{bmatrix} \end{bmatrix}$$

repräsentiert werden. Pfade werden in offensichtlicher Weise auf Untermatrizen ausgedehnt. Matrizen für NNF Formeln kann man wiederum mit Nichtklauseltableaux assoziieren und als eine platzsparende Notation für Tableaus auffassen. Genaueres zu allgemeinen Matrizen findet man in Bibel [1987].

5.6.2 Modellelimination (Loveland [1969])

Bei Modellelimination (ME) handelt es sich um eine Familie von Beweisprozeduren, bei deren Entwicklung folgende Ziele eine Rolle gespielt haben:

- kleiner Suchraum
- geringer Platzverbrauch
- mögliche Interpretation als systematisches Ausschließen von Modellen

Wir geben zunächst eine vereinfachte Version der ursprünglichen Formulierung (für AL):

Definition 5.23 *Eine* Kette *ist eine endliche Folge* $\langle L_1, \ldots, L_n \rangle$ *von Literalen. Jedes Literal ist entweder vom* Typ A *oder vom* Typ B *(Typ* A *Literale werden durch Umrandung gekennzeichnet).*

Sei S *eine Menge AL-Klauseln. Zu jeder Klausel aus* S *gehört eine Kette, die nur aus Typ* B *Literalen besteht.* ∎

$\boxed{A}$-Literale heißen auch **A̲ncestor-** bzw. **Vorgängerliterale**. B-Literale heißen **B̲lattliterale**. Der Grund für diese Benennung wird bald offensichtlich sein.

Definition 5.24 Zulässige Ketten *(über* S*) sind induktiv wie folgt definiert:*

Init: *Jede zu* $C \in S$ *gehörige Kette ist zulässig.*

Extend: *Ist* $\langle L_1, \ldots, L_n \rangle$ *eine zulässige Kette,* L_n *ein Typ* B *Literal, ist fernde* $\langle M_1, \ldots, M_m \rangle$ *die zu* $C \in S$ *gehörige Kette und* L_n *komplementär zu einem* M_j *(*$1 \leq j \leq m$*), dann ist*

$$\langle L_1, \ldots, L_{n-1}, \boxed{L_n}, M_1, \ldots, M_{j-1}, M_{j+1}, \ldots, M_m \rangle$$

eine zulässige Kette.

Reduce: *Ist* $K = \langle L_1, \ldots, L_n \rangle$ *eine zulässige Kette, seien* $\boxed{L_i}$ *ein Typ* A *und* $L_j, \ldots, L_n$ *Typ* B *Literale in* K *mit* $i < j$ *und* L_i *komplementär zu* L_j, *dann ist* $\langle L_1, \ldots, L_{j-1}, L_{j+1}, \ldots, L_n \rangle$ *eine zulässige Kette.*

Contract: *Ist* $K = \langle L_1, \ldots, \boxed{L_n} \rangle$ *eine zulässige Kette, seien* $\boxed{L_j}, \ldots, \boxed{L_n}$ *Typ* A *Literale, dann ist* $\langle L_1, \ldots, L_{j-1} \rangle$ *eine zulässige Kette.*

■

Definition 5.25 *Ein* Beweis mit Modellelimination *für* S *ist eine Folge von zulässigen Ketten über* S*, die* $\langle\rangle$ *enthält.* ■

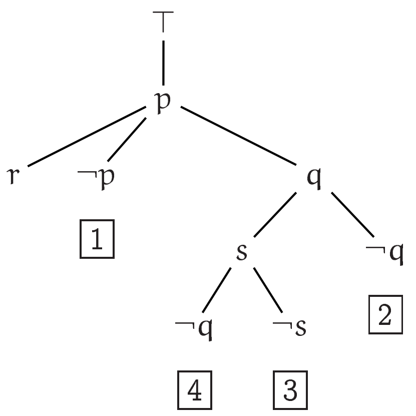
Satz 5.26 (Loveland [1968]) *Eine AL-Klauselmenge* S *ist unerfüllbar gdw es einen Beweis mit Modellelimination für* S *gibt.*

Die Beziehungen zwischen ME und Klauseltableaus können wie folgt zusammengefaßt werden:

- Typ B Literale entsprechen Blättern noch offener Zweige.
- $\boxed{A}$-Literale sind die Literale auf dem aktuellen Zweig, die kein Blatt sind.
- Die Initialisierungsregel der Modellelimination entspricht derjenigen der Konnektionstableaus.
- Die Extensionsregel entspricht einer Erweiterung des am weitesten rechts stehenden, offenen Tableauzweigs.
- Die Reduktionsregel entspricht dem Abschluß eines Zweigs.
- Die Kontraktionsregel kann nur angewandt werden, wenn das Ende der Kette ein A-Literal ist, d.h., wenn es keine offenen Zweige unter diesem Literal gibt. Das entspricht dem Löschen geschlossener Zweige.

Beispiel 5.27 $S = \{\{p\}, \{r, \neg p, q\}, \{s, \neg q\}, \{\neg q, \neg s\}, \{\neg q, \neg r\}, \{\neg r, q\}\}$

p	Init				
$\boxed{p}$	r	q	Ext		
$\boxed{p}$	r	$\boxed{q}$	s	Ext	
$\boxed{p}$	r	$\boxed{q}$	$\boxed{s}$	$\neg q$	Ext
$\boxed{p}$	r	$\boxed{q}$	$\boxed{s}$	Red	
$\boxed{p}$	r	Contr			
$\boxed{p}$	$\boxed{r}$	q	Ext		
$\boxed{p}$	$\boxed{r}$	$\boxed{q}$	$\neg r$	Ext	
$\boxed{p}$	$\boxed{r}$	$\boxed{q}$	Red		
Contr					



Der initiale Schritt kann mit einer beliebigen Klausel aus S erfolgen; wir wählen $\{p\}$. p ist nun zunächst ein B-Literal. Es muß also ein Extend-Schritt kommen, da p das einzige Literal in der Kette ist. Dieser erfolgt mit $\{\neg p, q, r\}$, wodurch p zum Typ A Literal wird. r und q sind neue B-Literale. Da kein Reduce-Schritt möglich ist, wird wieder erweitert, diesmal mit $\{s, \neg q\}$ (die verwendete Klausel muß $\neg q$ enthalten). Dann wird mit $\{\neg s, \neg q\}$ erweitert. Nun ist ein Reduce-Schritt möglich mit $\boxed{q}$. Dadurch wird das letzte Typ B Literal $\neg q$ gelöscht und die rechtsstehenden Typ A Literale müssen mit Contract entfernt werden. Nach zwei weiteren Extend-Schritten, einem Reduce-Schritt und einem Contract-Schritt hat man die leere Kette.

Das partielle Tableau auf der rechten Seite entspricht den ersten sechs Ketten in der ME Ableitung. Die zweite bis fünfte Kette entsprechen je einem Zweigabschluß, die im Tableau mit $\boxed{1}$ bis $\boxed{4}$ numeriert sind. ■

Modellelimination ist also nichts anderes als Konnektionstableaus mit Rightfirst-Auswahlregel. Die Vollständigkeit der Modellelimination ist damit ein Korollar von Satz 5.9.

Bemerkung 5.28 In den Originalarbeiten von Loveland ist die Interpretation der ME als Klauseltableaus nicht vorhanden. Diese wurde erst viel später durch Letz *et al.* [1992] nahegelegt.

Der Vollständigkeitsbeweis von Loveland ist wie auch die Definition der ME deutlich komplizierter als die Tableauversion.

In manchen Arbeiten wird ME mit zusätzlichen Einschränkungen versehen, die im Tableaurahmen als Regularität bzw. Bevorzugung von Abschlüssen vor Erweiterung leicht charakterisiert werden können.

Der hier vorgestellte ME Kalkül ist als „weak model elimination“ bekannt. Eine stärkere Variante von ME wird in Abschnitt 8.3.2 vorgestellt. ■

5.6.3 Modellgenerierung

Das Deduktionsverfahren **Modellgenerierung (MG)** wurde zuerst von Manthey and Bry [1988] vorgestellt. Die Idee dazu stammt aus dem Gebiet der *deduktiven Datenbanken*, woher auch die Einschränkung auf *bereichsbeschränkte*[2] Formeln kommt. Wie wir sehen werden, bewirkt diese Beschränkung, daß keine freien Variablen in einem Tableau vorkommen, was eine backtrackingfreie, parallele Implementierung ermöglicht. Eine solche wurde in sehr effizienter Weise durch Fujita and Hasegawa [1991] umgesetzt.

Klauseln werden wie in Abschnitt 3.3 als Regeln geschrieben, allerdings in umgekehrter Richtung:

$$p_1, \ldots, p_n \rightarrow q_1, \ldots, q_m$$

Definition 5.29 *Eine PL-Klauselmenge S ist* ***bereichsbeschränkt****, falls für alle*

$$C = p_1, \ldots, p_n \rightarrow q_1, \ldots, q_m \in S$$

gilt:

$$\mathit{Var}(\{p_1, \ldots, p_n\}) \supseteq \mathit{Var}(\{q_1, \ldots, q_m\}) \ .$$

■

Definition 5.30 *MG ist auf bereichsbeschränkten Formeln definiert. Folgende Regeln definieren die Menge $\mathcal{M}$ der* Modellkandidaten $M \subseteq At^0_\Sigma$ *für S:*

Init $\mathcal{M} = \{\emptyset\}$.

Modellerweiterung *Ist $C \rightarrow d_1, \ldots, d_k$ eine Klausel in S ($C = \emptyset$ möglich) und gibt es ein σ, so daß*

(a) $C\sigma \subseteq M \in \mathcal{M}$,

(a) $D\sigma \cap M = \emptyset$,

dann setze

$$\mathcal{M} := (\mathcal{M} - \{M\}) \cup \{M \cup \{d_1\sigma\}, \ldots, M \cup \{d_k\sigma\}\} \ .$$

Modell verwerfen *Ist $C \rightarrow$ eine Klausel in S ($C \neq \emptyset$) und gibt es ein σ, so daß*

$$C\sigma \subseteq M \in \mathcal{M} \ ,$$

dann setze

$$\mathcal{M} := \mathcal{M} - \{M\} \ .$$

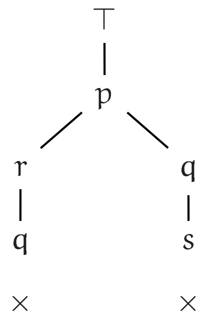
[2] Engl.: range-restricted.

■

Beispiel 5.31 Die Klauselmenge aus Beispiel 5.27 in MG Regelnotation lautet:

$$S = \{\rightarrow p \mid p \rightarrow r, q \mid q \rightarrow s \mid q, s \rightarrow \ \mid q, r \rightarrow \ \mid r \rightarrow q\}$$

Die systematische Konstruktion der Modellkandidaten für S kann durch folgenden Baum visualisiert werden.



■

Offensichtlich entsprechen die Modellkandidaten gerade Zweigen in Klauseltableaus. Das heißt, MG ist eine Variation der regulären schwachen Konnektionstableaus mit Neustart, wobei:

1. Neustartklauseln genau die positiven Klauseln sind und
2. alle negativen Literale der erweiternden Klausel ein Link mit dem erweiterten Zweig haben.

Daher findet man anstatt der Bezeichnung MG auch den Namen **positive Hypertableaus** in der Literatur.

Satz 5.32 *Eine bereichsbeschränkte PL-Klauselmenge S ist genau dann unerfüllbar, wenn die leere Menge von Modellkandidaten mit S hergeleitet werden kann.*

MG ist beweiskonfluent.

Beweis: Übung: man modifiziere den Beweis für geordnete Tableaus. ■

Die Bereichsbeschränktheit der Eingabeformeln impliziert, daß die Modellkandidaten aus Grundliteralen bestehen. Denn die erste Erweiterung muß mit einer positiven Klausel erfolgen. Wegen der Bereichsbeschränktheit sind positive Klauseln aber Grundklauseln. Ist $\mathcal{M}$ grund, bleibt es dies nach Erweiterung mit beliebigen Klauseln wegen der Bereichsbeschränktheit. Substitutionen werden also nie auf $\mathcal{M}$ angewandt.

`select_pair` und `select_mode` können damit deterministisch implementiert werden (Modell verwerfen wird bevorzugt). Falls nun `select_clause` (Modellerweiterung) fair ist, kann man eine Implementierung ohne BT erreichen.

`select_branch` ist meist `left_first` oder *parallel* implementiert. Da kein Zweig einen anderen beeinflußt, ist eine effiziente parallele Implementierung möglich (wenig Kommunikation notwendig). Für jeden neuen Modellkandidaten wird ein neuer Prozeß gestartet.

Kapitel 6

Verstärkungen von Klauseltableaus

6.1 Tableaus mit Schnitt

Tableauverfeinerungen schränken die Menge zulässiger Tableaus ein, um den Suchraum zu verkleinern. Ein entgegengesetzter Ansatz ist es, die Tableauregeln zu verstärken oder zu ergänzen, so daß kürzere Beweise als vorher möglich werden, die mit kleineren Schranken gefunden werden können.

Die Motivation hierzu ist das obige Beispiel der Klauselmengen Φ_n. Φ_n hat Klauseltableaubeweise, deren Größe polynomiell in $|\Phi_n|$ beschränkt ist.

Dies galt für die Menge $C_n \subset \Phi_n$ jedoch nicht. Wir wollen das noch einmal explizit festhalten.

Satz 6.1 *Jeder Klauseltableaubeweis für* C_n *besitzt* $n!\left(1+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{n!}\right)$ *oder mehr innere Knoten, aber die Größe von* C_n *ist nur* $n2^n$*.*

Beweis: Siehe D'Agostino [1992] oder Übung. ■

Man kann sich natürlich fragen, warum gerade diese Formelklasse Tableaubeweise polynomieller Größe haben sollte. Die Antwort ist, daß die Klauseln in C_n gerade durch die AL-Interpretationen über $\Sigma = \{p_1, \ldots, p_n\}$ widerlegt werden. Die Tautologien $\{\{p_i, \neg p_i\} \mid 1 \leq i \leq n\} = \Phi_n - C_n$ kodieren diese Interpretationen.

Klauseltableaus können aber AL-Interpretationen nicht effizient aufzählen. C_n stellt daher gewissermaßen eine Minimalanforderunge an die Stärke eines Kalküls dar, denn sogar Kalküle, die zunächst viel schwächer als Klauseltableaus aussehen, können mit C_n in polynomieller Zeit fertig werden:

Definition 6.2 *Sei* S *eine AL-Klauselmenge über* $\Sigma = \{p_1, \ldots, p_n\}$*. Die* Wahrheitstafel *für* S *ist eine Tabelle mit* $2^n + 2$ *Zeilen und* $n + 1$ *Spalten* *der folgenden Form:*

$$\begin{array}{cccc|c} & I_\Sigma & & & val_I(S) \\ p_1 & p_2 & \cdots & p_n & \\ \hline 0 & 0 & \cdots & 0 & val_\emptyset(S) \\ 0 & 0 & \cdots & 1 & val_{\{p_n\}}(S) \\ & & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 & val_\Sigma(S) \end{array}$$

■

Proposition 6.3 *Eine AL-Klauselmenge ist genau dann unerfüllbar, wenn in ihrer Wahrheitstafel nur* 0 *als Eintrag in der Spalte ganz rechts vorkommt.*

Eine AL-Klauselmenge ist genau dann allgemeingültig, wenn in ihrer Wahrheitstafel nur 1 *als Eintrag in der Spalte ganz rechts vorkommt.*

Beweis: Übung. ■

Für jedes S hat die Wahrheitstafel $2^{|\mathrm{Var}(S)|}(|\mathrm{Var}(S)|+1)$ Einträge. Alle Einträge können offenbar in linearer Zeit berechnet werden. Wegen $|\mathrm{Var}(C_n)| = n$ und $|C_n| = n2^n$ hat die Wahrheitstafelmethode also nur polynomiellen Aufwand für C_n.

Wie man an der Formelklasse C_n sieht, kann der Aufwand von Beweisen mit Klauseltableaus exponentiell höher sein als mit der simplen Wahrheitstafelmethode.

Da der Unterschied zwischen S_n und C_n nur in den atomaren Tautologien besteht, liegt folgende neue Tableauregel nahe:

$$\frac{}{p \mid \neg p} \qquad p \in \mathrm{At}_\Sigma \qquad \text{(Atomarer Schnitt)}$$

Die für den kurzen Beweis von C_n nötigen Tautologien können nun durch atomare Schnitte leicht erzeugt werden.

Die Bezeichnung „atomarer Schnitt“ leitet sich von der Verwandtschaft mit der Schnittregel des Gentzenkalküls bzw. Sequenzenkalküls ab. Dies ist ein wichtiger Punkt, auf den wir näher eingehen wollen:

Sequenzenkalkül für AL-Klauseln. Eine *Sequenz* ist ein Ausdruck der Gestalt $\Gamma \Rightarrow \Delta$, wobei Γ und Δ Tupel von Klauseln sind. $\Gamma \Rightarrow \Delta$ ist *allgemeingültig* gdw $\bigwedge_{G\in\Gamma} G \to \bigvee_{D\in\Delta} D$ allgemeingültig ist. Also ist eine Klauselmenge S unerfüllbar, falls

$$S \Rightarrow \emptyset \tag{6.1}$$

allgemeingültig ist. Anstatt $S \Rightarrow \emptyset$ schreiben wir $S \Rightarrow$.

Um die Allgemeingültigkeit von Formeln der Gestalt (6.1) nachzuweisen, reichen die folgenden Regeln (vgl. Vorlesung „Formale Systeme“) aus:

$$\frac{\Gamma, L_1, \Gamma' \Rightarrow \Delta \;\Big|\; \cdots \;\Big|\; \Gamma, L_n, \Gamma' \Rightarrow \Delta}{\Gamma, L_1 \vee \cdots \vee L_n, \Gamma' \Rightarrow \Delta} \tag{$\vee$-links}$$

$$\frac{\Gamma, \Gamma' \Rightarrow P, \Delta}{\Gamma, \neg P, \Gamma' \Rightarrow \Delta} \tag{$\neg$-links}$$

$$\frac{}{\Gamma, P, \Gamma' \Rightarrow \Delta, P, \Delta'} \tag{Axiom}$$

Ein Beweis von S im Sequenzenkalkül ist ein Baum mit der Wurzel $S \Rightarrow$, in dem neue Knoten mit den oben stehenden Regeln erzeugt wurden und in dem jeder Zweig mit einer Anwendung von (Axiom) endet.

Ein Beweis von C_2 zum Beispiel sieht im Sequenzenkalkül wie folgt aus:

$$\dfrac{\dfrac{\dfrac{p_1,p_1\vee\neg p_2,\neg p_1\vee p_2\Rightarrow p_1}{p_1,p_1\vee\neg p_2,\neg p_1\vee p_2,\neg p_1\Rightarrow}\quad\dfrac{\dfrac{p_1,p_1\vee\neg p_2,\neg p_2\Rightarrow p_1}{p_1,p_1\vee\neg p_2,\neg p_1,\neg p_2\Rightarrow}\quad\dfrac{p_1,p_1\vee\neg p_2,p_2\Rightarrow p_2}{p_1,p_1\vee\neg p_2,p_2,\neg p_2\Rightarrow}}{p_1,p_1\vee\neg p_2,\neg p_1\vee p_2,\neg p_2\Rightarrow}}{p_1,p_1\vee\neg p_2,\neg p_1\vee p_2,\neg p_1\vee\neg p_2\Rightarrow}\qquad\vdots}{C_2\Rightarrow}$$

Während die logische Metastruktur eines Klauseltableaus einer DNF über den Literalen in den Knoten entspricht, kann man einen Sequenzenbeweis als KNF über die Elemente der Sequenzen betrachten, wobei aus jedem Zweig genau eine Sequenz genommen wird. Axiome des Sequenzenkalküls entsprechen tautologischen Klauseln, wegen der Dualität also komplementären Literalen. Die Dualität zwischen den Kalkülen läßt sich wie folgt erweitern:

- Sequenzen entsprechen (partiellen) Tableauzweigen;
- Anwendung von ($\vee$-links) entspricht einer Tableauerweiterung;
- Atome in Γ entsprechen positiven Literalen in einem Tableau;
- Atome in Δ entsprechen negativen Literalen in einem Tableau;
- Axiome entsprechen einem Zweigabschluß.

In Sequenzenkalkülen gibt es außerdem noch die sogenannte *Schnittregel:*

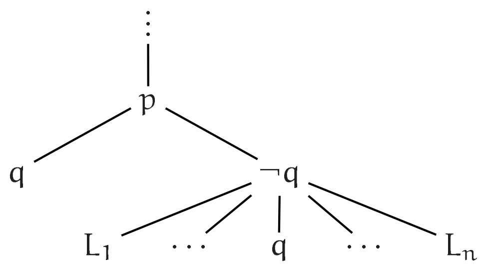
$$\frac{\Gamma, \Gamma' \Rightarrow \Delta, \phi, \Delta' \mid \Gamma, \phi, \Gamma' \Rightarrow \Delta, \Delta'}{\Gamma, \Gamma' \Rightarrow \Delta, \Delta'} \qquad \text{(Cut)}$$

Dabei ist die *Schnittformel* ϕ eine atomare Formel.[1] Damit und aufgrund der Dualität zwischen Sequenzen- und Tableaukalkül ist klar, daß die Schnittregel in jedem Fall dasselbe bedeutet.

Die Korrektheit der Schnittregel wird im Zusammenhang mit Sequenzensystemen bewiesen, ist aber auch leicht einzusehen: angenommen $\Gamma, \Gamma' \Rightarrow \Delta, \phi, \Delta'$ und $\Gamma, \phi, \Gamma' \Rightarrow \Delta, \Delta'$ werden von I gleichzeitig erfüllt. Dann muß auch $\Gamma, \Gamma' \Rightarrow \Delta, \Delta'$ von I erfüllt werden, denn es ist entweder $I(\phi) = F$ oder $I(\overline{\phi}) = F$ und damit wäre dann auch eine der Voraussetzungen nicht erfüllt. Die umgekehrte Richtung ist trivial, da jedes Modell von A auch Modell von $A \vee B$ ist.

Der Nachteil der Schnittregel ist, daß ihre Anwendung völlig unbeschränkt ist, das heißt, man kann beliebige Atome einsetzen, deren Nützlichkeit zunächst unklar ist.

Für Klauseltableaus wird man sie daher einschränken: die atomare Schnittregel darf nur angewandt werden, wenn die Schnittformel p oder ihr Negat $\neg p$ ein komplementäres Paar mit einem Literal L der Klausel, mit welcher der aktuelle Zweig als nächstes erweitert wird, bildet:

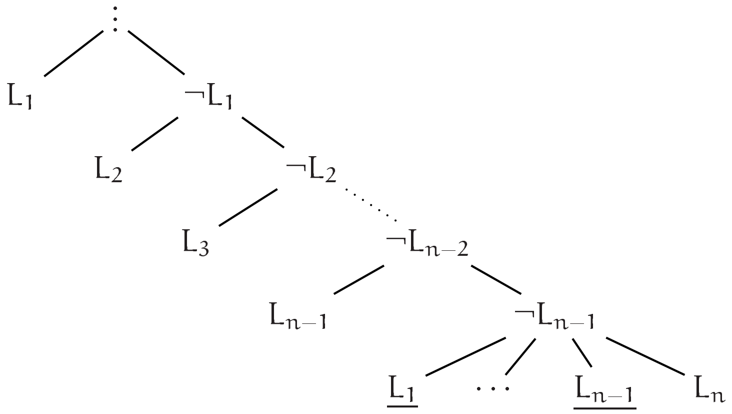


Dieses Vorgehen hat den Vorteil, daß nicht mehr offene Zweige erzeugt werden als bei einer Standardregelanwendung (da mindestens ein neuer Zweig sofort geschlossen werden kann), und sie schließt keine kürzeren Beweise gegenüber der unbeschränkten Version aus, denn wenn die Einschränkung nicht erfüllt ist, dann kann die Schnittregel auch den Beweis nicht verkürzen.

[1] Diese, wie auch viele andere Festlegungen sind für den vorliegenden Kontext (Klausellogik) vereinfacht worden und dürfen nicht als allgemeine Einschränkungen von Sequenzenkalkülen aufgefaßt werden. Zu allgemeinen Sequenzenkalkülen informiert ausführlich zum Beispiel Gallier [1986].

6.2 Tableaus mit lokalen Lemmata

Wendet man vor einer Tableauerweiterung mit $C = \{L_1, \dots, L_n\} \in S$ die atomare Schnittregel $(n-1)$-mal auf $L_1, \dots, L_{n-1}$ an, so ergibt sich (schreibe $\neg L$ anstatt $\overline{L}$ wegen der Lesbarkeit):



Nach Löschung der geschlossenen Zweige können die insgesamt n Regelanwendungen wie folgt zusammengefaßt werden:

$$\begin{array}{c|r|r|c|r}
\hline
L_1 & \neg L_1 & \neg L_1 & \cdots & \neg L_1 \\
 & L_2 & \neg L_2 & \cdots & \neg L_2 \\
 & & L_3 & \cdots & \neg L_3 \\
 & & & & \vdots \\
 & & & \ddots & \neg L_{n-1} \\
 & & & & L_n
\end{array}$$

$$\{L_1, \dots, L_n\} \in S$$

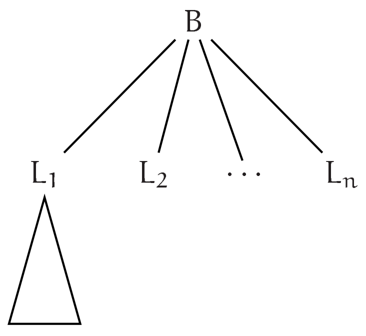
Wird die normale Tableauerweiterungsregel durch die obenstehende Regel ersetzt, so nennen wir die daraus resultierende Beweisprozedur *Klauseltableaus mit lokalen Lemmata*. Sie sind *lokal*, da die „Lemmata" $\neg L_i$ an einen Zweig gebunden sind.

Tableaus mit Lemmata sind somit ein Spezialfall von Tableaus mit atomarem Schnitt. Umgekehrt sind Tableaus mit atomarem Schnitt (unter der genannten Einschränkung) gerade Tableaus mit Lemmata, wobei einige Formeln ignoriert werden.

Da Klauseltableaus mit (eingeschränktem) Schnitt C_n polynomiell lösen können, Klauseltableaus ohne Schnitt jedoch nicht, folgt, daß Lemmagenerierung den Kalkül „stärker" macht. Dies wird in Kürze noch präzisiert.

Die negierten Literale in der neuen Erweiterungsregel heißen *Lemmata*. Diese Bezeichnung wollen wir kurz begründen:

Sei B der aktuelle Zweig, der mit der Klausel $C = \{L_1, \dots, L_n\} \in S$ erweitert wird. Jeder der neuen Zweige $B \cup \{L_1\}, \dots, B \cup \{L_n\}$ muß geschlossen werden, insbesondere $B \cup \{L_1\}$. Nach dem Abschluß von $B \cup \{L_1\}$



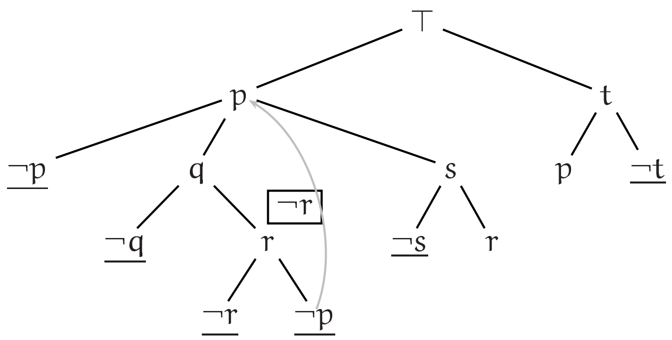
ist gezeigt, daß $S \cup B \cup \{L_1\}$ unerfüllbar ist (folgt aus der Korrektheit von Klauseltableaus) und daraus folgt $S \cup B \models \neg L_1$. Es ist also korrekt, das Literal $\neg L_1$ zu allen anderen Zweigen hinzuzufügen. Durch Induktion erhält man die Erweiterungsregel mit Lemmata.

Die Benennung ist nun klar: die Teiltableaus unterhalb der L_i können als Beweise der Lemmata $S \cup B \models \neg L_i$ angesehen werden.

Man beachte daß es für jede Anordnung der Literale in C einer andere Erweiterungsregel mit Lemmata gibt, also $n!$ verschiedene Regeln mit Lemmata pro Klausel der Länge n. Für die indeterministische Komplexität (Länge des kleinsten Beweises) ist dies unwesentlich. In einer deterministischen Implementierung aber nicht. Eine Verbesserung wird durch die Kalkülvariante erzielt, die im nächsten Abschnitt beschrieben ist. Man könnte sie als „lazy lemma generation" bezeichnen, da man sie als die gezielte Erzeugung von Lemmata interpretieren kann, nachdem Informationen über geschlossene Unterbäume ausgewertet wurden.

6.3 Klauseltableaus mit Hochklappen

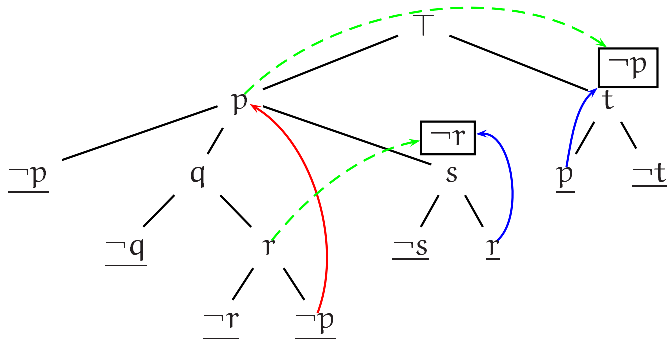
$S = \{\{p,t\},\{p,\neg t\},\{\neg p,q,s\}\{\neg q,r\},\{\neg r,\neg p\},\{\neg s,r\}\}$



Nach Abschluß von $B = \{p, q, r\}$ weiß man, daß $S \cup \{p, q\} \models \neg r$. Dieses Lemma bringt jedoch nichts ein, da der linke Nachbarzweig schon geschlossen wurde. Es muß aber sogar $S \cup \{p\} \models \neg r$ gelten, da q nicht für Abschluß von B verwendet wurde. Dies legt folgende Verstärkung der Tableaus mit Lemmata nahe:

Tableau mit Hochklappen: In Klauseltableaus mit Lemmata dürfen die Lemmata bis zum niedrigsten für ihren Beweis verwendeten Literal hochgeklappt werden Letz *et al.* [1992]; Letz [1993] und in jedem Zweig unterhalb dieser Position benutzt werden. Das Beispiel von eben nun noch einmal mit Hochklappen:

$S = \{\{p, t\}, \{p, \neg t\}, \{\neg p, q, s\}\{\neg q, r\}, \{\neg r, \neg p\}, \{\neg s, r\}\}$



Die durchgezogenen Pfeile indizieren Abschlüsse, die gestrichelten Pfeile zeigen, wie Lemmata verschoben wurden. Das Tableau ist nun ohne weiteres geschlossen.

Letz [1993] zeigte, daß Hochklappen den Kalkül nicht wirklich stärker macht, denn durch eine geschickte Lemmata- und Klauselwahl kann derselbe Effekt wie durch Hochklappen erzielt werden. Der Vorteil des Hochklappens ist also eher, daß eine falsche Entscheidung in `select_clause` ohne teures Backtracking korrigiert werden kann. Dies ist in der Prädikatenlogik besonders wichtig.

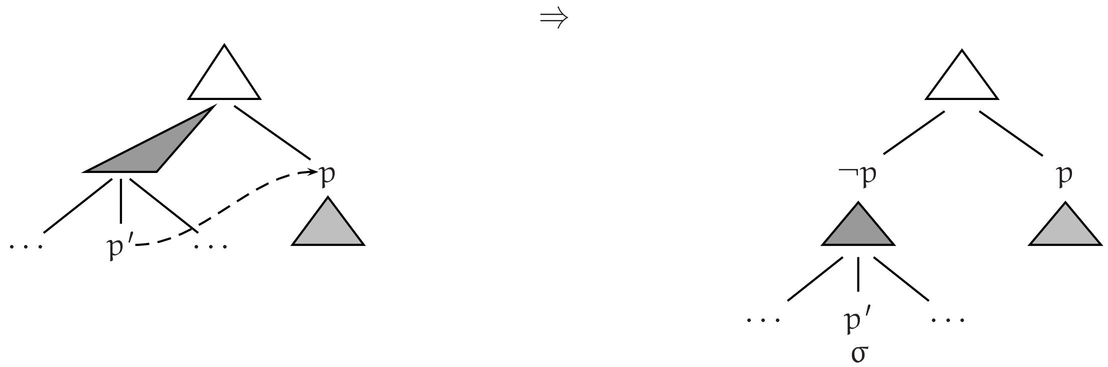
6.4 Tableaus mit Faktorisierung

Seien B, B' Zweige in einem Tableaus, so daß $B \subseteq B'\sigma$. Falls nun B geschlossen ist, kann auch B' ohne weiteres **geschlossen werden mit Verweis auf** B. Die gegenseitigen Verweise dürfen allerdings nicht zyklisch sein.

Die Rechtfertigung für dieses Vorgehen ist wie folgt: Sei $S \cup B$ unerfüllbar. Wegen $B \subseteq B'\sigma$ gilt $S \cup B \subseteq S \cup B'\sigma$. Also ist $S \cup B'\sigma$ unerfüllbar, damit ist $B'\sigma$ geschlossen und es kann auch B' mit einer geeigneten Substitution geschlossen werden.

Der aus der zusätzlichen Abschlußregel resultierende Kalkül, genannt **Tableaus mit Faktorisierung**, erweist sich als Spezialfall von Tableaus mit Lemmata.

$p = p'\sigma$ komm im linken Zweig o.B.d.A. als Blatt vor. Folgendes Schema wird verwendet, um Tableaus mit Faktorisierung auf Tableaus mit Lemmata zu reduzieren:



Im linken Tableau ist der Ast B' durch einen Verweis von p' auf p geschlossen. Im rechten Tableau taucht $\neg p$ als Lemma auf.

6.5 Nachteile der Lemmagenerierung

In der AL ist Lemmata immer von Vorteil. In der Prädikatenlogik aber können Lemmata den Suchraum durch zusätzliche Abschlußmöglichkeiten so stark aufblähen, daß der Vorteil, daß kürzere Beweise möglich sind, völlig neutralisiert wird.

Beispiel 6.4 Betrachte das Konnektionstableau mit Lemmata (die Lemmata sind eingerahmt) in Abbildung 6.1 für die Klauselmenge

$S = \{\{\neg p(x), q(x), r(x)\}, \{p(a)\}, \{\neg q(a)\}, \{p(b)\}, \{\neg q(b)\}, \{\neg r(b)\}\}$.

Im linken Zweig wurde mit $p(a)$ geschlossen anstatt mit $p(b)$, was richtig gewesen wäre. Ohne die Lemmata könnte im rechten Zweig daraufhin keine Erweiterung erfolgen und es würde sofort Backtracking zur richtigen Alternative initiiert. Mit dem Lemma $p(X)$ ist aber ein Erweiterungsschritt möglich, und der Fehler setzt sich noch lange fort.

Für dies spezielle Beispiel würde zwar Regularität ($r(a)$ ist irregulär auf dem rechten Zweig) oder die Einschränkung von Erweiterungsschritten auf Nichtlemmata helfen, es läßt sich jedoch leicht verallgemeinern (Übung). ■

Auf der anderen Seite ist Lemmaverwendung, da es eine reine AL-Regel ist, für viele prädikatenlogische Probleme noch nicht so stark, wie man es gerne hätte. Betrachten wir zum Beispiel das partielle Klauseltableau in Abbildung 4.1. Lemmata mit Hochklappen würde das Lemma $\neg q(c)$ bis zur Wurzel

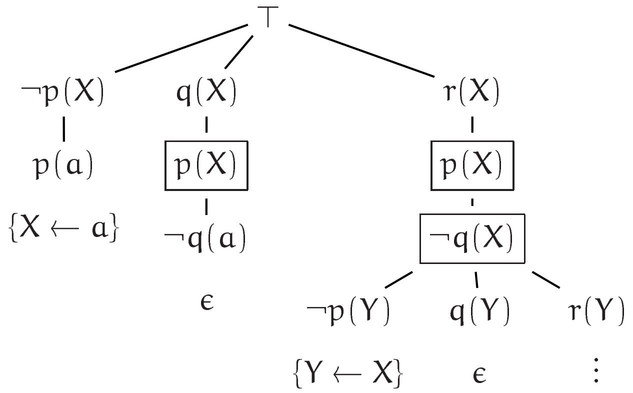


Abb. 6.1: Konnektionstableau mit Lemmata, das den größeren Suchraum bei Lemmaverwendung veranschaulicht.

des Tableaus hochschieben. Dennoch kann der noch offene Zweig nicht damit geschlossen werden, weil darin $q(x)$ in einer anderen Instanz benötigt wird.

Es ist deshalb wichtig zu bemerken, daß in dem Untertableau unterhalb von $q(X_2)$ nicht nur das Lemma $\neg q(c)$ bewiesen wurde, sondern sogar die Generalisierung $\forall x \neg q(x)$. Dies ist immer dann der Fall, wenn beim Abschluß des betreffenden Untertableau keine Variablen, die oberhalb des Lemmas auftauchen, belegt werden.

Wenn man $\forall x \neg q(x)$ als Lemma an der Wurzel des Tableaus zur Verfügung hat, kann der rechte Zweig sofort geschlossen werden.

Man beachte, daß die Implementierung einer erweiterten Hochklappregel, wie soeben skizziert, die Kenntnis der in jedem Untertableau belegten Variablen erfordert, was u.U. recht kostspielig sein kann. Die Probleme mit einem vergrößerten Suchraum werden außerdem noch verschärft.

Kapitel 7

Resolution

7.1 Grundlagen

Resolutionskalküle haben die Forschung im Gebiet „Automatisches Beweisen" bis in die 80er Jahre dominiert. Der Hauptgrund: Unifikation war von vorne herein (d.h., ab etwa 1963) ein zentraler Bestandteil der PL-Resolution.

Die Verwendung von Unifikation ist absolut unverzichtbar, um den Suchraum beim Auffinden der dem Satz von Herbrand genügenden Instanzen von Klauseln einzuschränken. Der systematische Einsatz von Unifikation bei der Implementierung von Tableaukalkülen aber begann erst etwa 1985 (bei der Konnektionsmethode und Matings zwar schon Mitte der siebziger Jahre, aber dort wurde wenig ernsthafte Implementierungsarbeit betrieben).

Ein weiterer Vorteil von Resolution ist, daß durch die Einschränkung auf SNF im wesentlichen eine einzige Inferenzregel für eine vollständige Beweisprozedur genügt, was effiziente Implementierungen erleichtert. Wie wir sehen werden, kann bei der Implementierung der meisten Resolutionsverfeinerungen auch auf Backtracking verzichtet werden, da Resolution beweiskonfluent ist und eine Fairneß leicht zu erreichen.

Definition 7.1 *Seien* $C = \{L, L_1, \ldots, L_n\}$ *und* $D = \{\overline{L}, K_1, \ldots, K_m\}$ *AL-Klauseln. Dann heißt* $E = (C - \{L\}) \cup (D - \{\overline{L}\})$ AL-Resolvente *von* C *und* D*.* C *und* D *heißen* Elternklauseln *von* E*,* L *ist das* resolvierte Literal. ■

Beispiel 7.2 Sei $C = \{p, q, \neg r\}$ und $D = \{q, s, r\}$. Mit resolviertem Literal $L = \neg r$ ist $\{p, q, s\}$ eine AL-Resolvente von C und D. Man beachte, daß Klauseln Mengen sind, das heißt, es wird kein Unterschied zwischen $\{q\}$ und $\{q, q\}$ gemacht. ■

Sei o.B.d.A. L atomar. Wenn man die Resolutionsregel in der Notation der Sequenzenkalküle – also so:

$$\frac{\overline{L_1}, \ldots, \overline{L_n} \Rightarrow L \;\big|\; L \Rightarrow K_1, \ldots, K_m}{\overline{L_1}, \ldots, \overline{L_n} \Rightarrow K_1, \ldots, K_m}$$

– dann wird klar, daß sie nichts anderes als die Schnittregel ist, denn es gilt die Kette von Äquivalenzen:

$$
\begin{aligned}
\left(\overline{L_1}, \ldots, \overline{L_n} \Rightarrow K_1, \ldots, K_m\right) \quad &\Leftrightarrow \quad \left(\bigwedge_{i=1}^{n} \overline{L_i} \rightarrow \bigvee_{j=1}^{m} K_j\right) \\
&\Leftrightarrow \quad \left(\bigvee_{i=1}^{n} \overline{\overline{L_i}} \vee \bigvee_{j=1}^{m} K_j\right) \\
&\Leftrightarrow \quad \left(\bigvee_{i=1}^{n} L_i \vee \bigvee_{j=1}^{m} K_j\right) \\
&\Leftrightarrow \quad \left((C - \{L\}) \cup (D - \{\overline{L}\})\right)
\end{aligned}
$$

Damit ist klar:

Lemma 7.3 *Sei* E *eine AL-Resolvente von Klauseln* C *und* D. *Ist* I *ein Modell von* $\{C, D\}$ *dann auch von* E *und umgekehrt.*

Obgleich Resolution logisch dasselbe wie die Schnittregel ist, gibt es entscheidende Unterschiede in der Anwendung:

1. Die Resolutionsregel wird „von oben nach unten“ angewandt, d.h., die Schnittformel ist immer bekannt.

2. Dementsprechend muß eine Resolutionsableitung nicht die Baumstruktur eines Sequenzen- bzw. Tableaubeweises besitzen, sondern sie kann ein DAG (gerichteter azyklischer Graph) sein.[1]

In der PL muß, wie beim Abschluß von Zweigen im Tableaukalkül, vor einem Resolutionsschritt durch Anwendung einer geeigneten Substitution auf die Elternklauseln ein komplementäres Paar erzeugt werden.

Definition 7.4 *Seien* $C = \{L, L_1, \ldots, L_n\}$ *und* $D = \{M, K_1, \ldots, K_m\}$ *PL-Klauseln, so daß Var*$(C) \cap$ *Var*$(D) = \emptyset$ *und es einen mgu* σ *von* L *und* $\overline{M}$ *gibt. Dann heißt*

$$(C - \{L\})\sigma \cup (D - \{M\})\sigma$$

PL-Resolvente *von* C *und* D. ■

[1] Man könnte natürlich auch in einem Tableau gemeinsame Unterbäume miteinander identifizieren, aber da in Tableaus komplexe Formeln *analysiert* werden (im Gegensatz zu Resolution, die Resolventen *synthetisiert*) kann dies teils erst *a posteriori* erkannt werden. Eine weitere Schwierigkeit ist, daß Substitutionen global auf ein Tableau einwirken, aber nur lokal auf eine Resolvente.

Dies ist genau die Resolutionsregel, wie sie von Robinson [1965b] eingeführt wurde.

Beispiel 7.5 $C = \{\overbrace{p(a,x)}^{l}, \overbrace{p(x,x)}^{C-\{l\}}\}$, $D = \{\overbrace{\neg p(y,y)}^{l'}\}$.

C und D besitzen die Resolventen $E = (C - \{l\})\sigma \cup (C' - \{l'\})\sigma = \{p(x,x)\}\sigma \cup \{\}\sigma = \{p(a,a)\}$ und $\{p(a,x)\}$, wobei $\sigma = \{y \leftarrow a,\ x \leftarrow a\}$. ■

In der PL gibt es einen weiteren, entscheidenden Unterschied zwischen Resolution und Tableaukalkülen:

3. Alle Eingabeklauseln *und* alle Resolventen sind universell bzgl. den in ihnen vorkommenden Variablen (im Gegensatz zu den in einem Tableau vorkommenden Literalen).

Insbesondere dürfen – und müssen – die Variablen in den Elternklauseln vor dem Resolutionsschritt gebunden umbenannt werden.

Beispiel 7.6 $\{p(x)\}$ und $\{\neg p(f(x)), q(x)\}$ besitzen für keine Substitution ein komplementäres Paar, die äquivalenten Klauseln $\{p(y)\}$ und $\{\neg p(f(x)), q(x)\}$ aber schon. Dies verdeutlicht die Notwendigkeit der Umbenennung. ■

In der Literatur Chang and Lee [1973] findet man manchmal auch die folgende Variante der PL-Resolutionsregel:

$$(C\sigma - \{L\}\sigma) \cup (D\sigma - \{\overline{L}\}\sigma)$$

Für obiges Beispiel ergibt sich damit:

$$\begin{aligned} E &= (C\sigma - \{L\}\sigma) \cup (D\sigma - \{\overline{L}\}\sigma) \\ &= (\{p(a,a)\} - \{p(a,a)\}) \cup (\{\neg p(a,a)\} - \{\neg p(a,a)\}) \\ &= \square \end{aligned}$$

Dies entspricht Substitution (mit σ) und nachfolgender AL-Resolution. Wir verwenden immer die ursprüngliche Definition.

Notation: Sei $S = \{C_1, \ldots, C_m\}$ eine Menge von PL-Klauseln. Wir verwenden $R(C_i, k; C_j, l)$ als Bezeichnung für die Resolvente mit den Elternklauseln C_i und C_j, wobei über das k-te Literal in C_i und das l-te Literal in C_j resolviert wurde.

Lemma 7.7 *Sei S eine Menge von PL-Klauseln, die ein Modell M besitzt, und E eine PL-Resolvente aus $C, D \in S$. Dann ist M auch ein Modell von $S \cup \{E\}$.*

Beweis: Es genügt, das Lemma für Herbrand-Modelle zu zeigen.

In Herbrand-Interpretationen besteht das Universum aus den Grundtermen der Formel, also sind Variablenbelegungen β gleichzeitig Grundsubstitutionen β. Das Substitutionstheorem vereinfacht sich daher zu:

Für Herbrand-Interpretationen $\mathcal{D}$, Variablenbelegungen β, Terme t und Formeln A gilt:

$$\mathrm{val}_{\mathcal{D},\beta}(t) = \mathrm{val}_{\mathcal{D},\beta}(t\beta) = t\beta \text{ und } \mathrm{val}_{\mathcal{D},\beta}(A) = \mathrm{val}_{\mathcal{D},\beta}(A\beta) \tag{*}$$

Außerdem gilt (in allen Interpretationen) für Grundterme t und beliebige Belegungen β, β':

$$\mathrm{val}_{\mathcal{D},\beta}(t) = \mathrm{val}_{\mathcal{D},\beta'}(t) \tag{**}$$

Laut Annahme gibt es ein Herbrand-Modell $\mathcal{D}$ von S. Wir nehmen o.B.d.A. an, daß E variablendisjunkt mit S ist. Sei β beliebig, $L \in C$ sei das resolvierte Literal und σ der mgu von L und $\overline{L}$.

1. Fall: Sei $\mathrm{val}_{\mathcal{D},\beta}(L\sigma) = W$; dann folgt, daß

$$\begin{aligned} F &= \mathrm{val}_{\mathcal{D},\beta}(\overline{L\sigma}) \\ &= \mathrm{val}_{\mathcal{D},\beta}(\overline{L}\sigma) \\ &\overset{(*)}{=} \mathrm{val}_{\mathcal{D},\beta}(\overline{L}\sigma\beta) \\ &\overset{(**)}{=} \mathrm{val}_{\mathcal{D},\beta\circ\sigma}(\overline{L}\sigma\beta) \\ &\overset{(*)}{=} \mathrm{val}_{\mathcal{D},\beta\circ\sigma}(\overline{L}) \ . \end{aligned}$$

Da nach Voraussetzung $\mathrm{val}_{\mathcal{D},\beta\circ\sigma}(D) = W$ (denn D hat als Modell $\mathcal{D}$, muß also für alle Belegungen zu W auswerten), muß auch $\mathrm{val}_{\mathcal{D},\beta\circ\sigma}(D - \{\overline{L}\}) = W$ gelten, und ähnlich wie oben folgt, daß dann $(D - \{\overline{L}\})\sigma$, und damit E von $\mathcal{D}$ und β erfüllt wird.

2. Fall: Sei $\mathrm{val}_{\mathcal{D},\beta}(L\sigma) = F$; dann gilt

$$\mathrm{val}_{\mathcal{D},\beta}(\overline{L}\sigma) = W$$

und wir schließen ähnlich wie oben.

β war beliebig gewählt, also ist $\mathcal{D}$ ein Modell von E. ■

Die Resolutionsregel in der angegebenen Form ist für Prädikatenlogik *nicht* vollständig.

Beispiel 7.8 $S = \{\{p(x), p(y)\}, \{\neg p(u), \neg p(v)\}\}$ ist offenbar unerfüllbar, aber es können nur Resolventen der Länge 2 daraus generiert werden. ■

Definition 7.9 *Sei* C *eine PL-Klausel,* $A \subseteq C$ *und* σ *ein mgu von* A*; dann heißt* $C\sigma$ Faktor *von* C*. Falls* $|C\sigma| < |C|$ *ist* $C\sigma$ *ein* nicht-trivialer Faktor *von* C*.* ∎

Beispiel 7.10 $\{p(x)\} = \{p(x), p(y)\}\{y \leftarrow x\}$ ist ein nicht-trivialer Faktor von $\{p(x), p(y)\}$. ∎

Notation: Sei $S = \{C_1, \ldots, C_m\}$ eine Menge von PL-Klauseln. Wir verwenden $F(C_i, k_1, \ldots, k_m)$ als Bezeichnung für den Faktor $C_i\sigma$, wobei σ der mgu des k_1-ten bis k_m-ten Literals von C_i ist.

Lemma 7.11 *Sei* S *eine Menge von PL-Klauseln, die ein Modell besitzt, und* D *ein Faktor einer Klausel* $C \in S$*. Dann besitzt* $S \cup \{D\}$ *dasselbe Modell wie* S*.*

Beweis: Ähnlich wie Lemma 7.7 nur einfacher: Übung. ∎

Es wird sich als günstig erweisen, die Faktorisierungsregel und die PL-Resolutionsregel in eine einzige Regel zusammenzuziehen:

Definition 7.12 *Seien* C *und* D *PL-Klauseln, so daß* $Var(C) \cap Var(D) = \emptyset$*, und seien* $C' = \{L, L_1, \ldots, L_n\}$ *und* $D' = \{M, K_1, \ldots, K_m\}$ *Faktoren von* C *und* D*. Weiterhin gebe es einen mgu* σ *von* L *und* $\overline{M}$*. Dann heißt*

$$(C' - \{L\})\sigma \cup (D' - \{M\})\sigma$$

PL-Resolvente mit Faktorisierung *von* C *und* D*.* ∎

Notation: $RF(C_i, k_1, \ldots, k_q; C_j, l_1, \ldots, l_r)$ analog zur PL-Resolution ohne Faktorisierung.

Beispiel 7.13 Seien C und D wie in Beispiel 7.8. Mittels PL-Resolution mit Faktorisierung kann nun in einem Schritt $RF(C, 1, 2; D, 1, 2)$ die Resolvente $\square$ gebildet werden. Die verwendeten Faktoren sind $C' = \{p(x)\}$ und $D' = \{\neg p(u)\}$.
∎

Aus den Lemmata 7.7 und 7.11 folgt man sofort:

Lemma 7.14 *Sei* S *eine Menge von PL-Klauseln, die ein Modell* M *besitzt, und sei* E *eine PL-Resolvente mit Faktorisierung von* $C, D \in S$*. Dann ist* M *auch Modell von* $S \cup \{E\}$*.*

Definition 7.15 *Sei* S *eine Menge von PL-Klauseln.* C *ist aus* S *durch* Resolution *abgeleitet, in Symbolen* $S \vdash_R C$*, falls es eine Folge von Klauseln* $\langle C_1, \ldots, C_n \rangle$ *gibt, so daß* $C_n = C$ *und für alle* C_i*,* $1 \le i \le n$ *gilt:*

1. $C_i \in S$ *oder*

2. C_i *ist eine PL-Resolvente mit Faktorisierung von bereinigten Klauseln* C'_j, C'_k, *die durch gebundene Umbenennung aus* C_j *und* C_k *mit* $j, k < i$ *hervorgehen.*

$\langle C_1, \ldots, C_n \rangle$ *heißt* (Resolutions-)Ableitung *von* C *aus* S *und* n *ist die* Länge *der Ableitung.* ■

In völlig analoger Weise ist die Relation $\vdash_{RA}$ für AL-Klauselmengen definiert, wobei nur die AL-Resolutionsregel (natürlich ohne Umbenennung) verwendet werden darf.

Beispiel 7.16 $S_n = \{\{p(0)\}, \{\neg p(x), p(s(x))\}, \{\neg p(s^{2^n}(x))\}\}$.
Eine Resolutionsableitung von $\square$ aus S ist wie folgt:

$$\begin{array}{lll}
C_1: & \{p(0)\} & S \\
C_2: & \{\neg p(x), p(s(x))\} & S \\
C_3: & \{\neg p(s^{2^n}(x))\} & S \\
C_4: & \{\neg p(x), p(s^2(x))\} & R(C_2, 1; C_2, 2) \\
\vdots & \vdots & \vdots \\
C_{i+3}: & \{\neg p(x), p(s^{2^i}(x))\} & R(C_{i+2}, 1; C_{i+2}, 2) \\
C_{n+3}: & \{\neg p(x), p(s^{2^n}(x))\} & R(C_{n+2}, 1; C_{n+2}, 2) \\
C_{n+4}: & \{p(s^{2^n}(0))\} & R(C_1, 1; C_{n+3}, 1) \\
C_{n+5}: & \square & R(C_3, 1; C_{n+4}, 1)
\end{array}$$

Die Länge der Ableitung ist $n + 5$. Man beachte, daß fast alle Klauseln mit zwei Varianten von sich selbst resolvieren. Zum Beispiel gilt

$$\begin{array}{rcl}
C_2 & = & C = \{\neg p(x), \overbrace{p(s(x))}^{l}\} \\
C_2 & = & C' = \{\overbrace{\neg p(x')}^{l'}, p(s(x'))\} \\
\sigma & = & \{x' \leftarrow s(x)\}
\end{array}$$

und also

$$C_4 = E = \{\neg p(x), p(s(s((x)))\} \ .$$

Die Universalität von C_2 bzgl. x ist dabei entscheidend.

Die gleiche Ableitung kann in DAG-Notation angegeben werden, um die Abhängigkeiten der Resolventen besser zu veranschaulichen. Die Knoten des zu einer Resolutionsableitung gehörenden DAG sind $\{C \mid C \text{ kommt in } \Gamma \text{ vor}\}$,

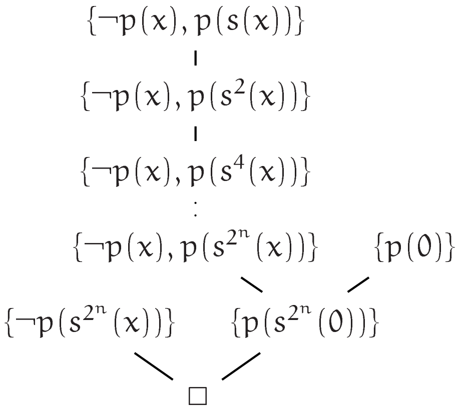


Abb. 7.1: DAG-Darstellung der Resolutionsableitung aus Beispiel 7.16.

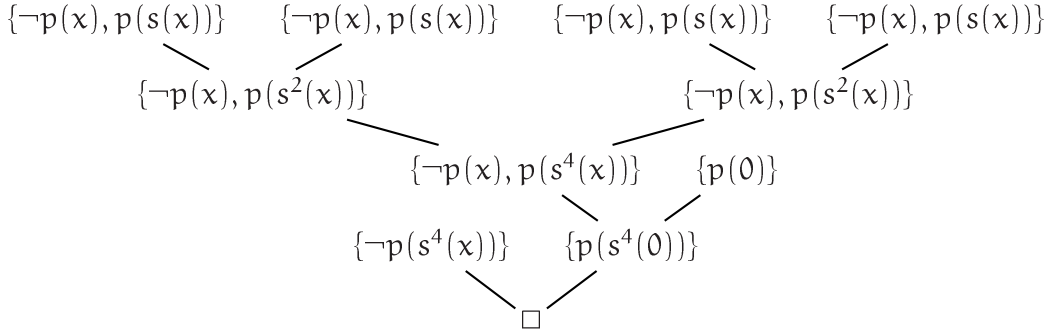


Abb. 7.2: Baumdarstellung der Resolutionsableitung aus Beispiel 7.16.

die Kanten sind $\{\langle C, E\rangle \mid C \text{ Elternklausel von } E\}$. Für das Beispiel ergibt dies den in Abbildung 7.1 dargestellten DAG.

Aus einem DAG kann man wie üblich einen Baum machen, was allerdings exponentiellen Platz brauchen kann. Das aktuelle Beispiel zeigt dies. Die obige Resolutionsableitung als Baum hingeschrieben für $n = 2$ ist in Abbildung 7.2 dargestellt (alle Elternklauseln werden implizit umbenannt).

Offensichtlich müssen im allgemeinen Fall insgesamt 2^n Kopien der Klausel $\{\neg p(x), p(s(x))\}$ angelegt werden. ■

Eine Resolutionsableitung in Baumnotation kann also exponentiell mehr Platz verbrauchen als in DAG- oder in linearer Notation.

Satz 7.17 *Sei S eine PL-Klauselmenge. Wenn $S \vdash_R \square$ dann ist S unerfüllbar.*

Beweis: Angenommen S wäre erfüllbar, hätte also ein Modell. Wir zeigen per Induktion über die Länge der Ableitung, daß für alle C gilt: wenn $S \vdash_R C$, dann hat $S \cup \{C\}$ dasselbe Modell.

$n = 1$: Es muß $C \in S$ gelten, also $S = S \cup \{C\}$.

$n > 1$: Sei $\langle C_1, \ldots, C_n \rangle$ eine Ableitung von C aus S. Es gibt entsprechend der Definition zwei Fälle: 1. $C \in S$, s.o. 2. $C = RF(C_i', \vec{k}; C_j', \vec{l})$ für gewisse $\vec{k}$, $\vec{l}$ und $i, j < n$. Dann haben C_i, C_i', C_j und C_j' dasselbe Modell wie S nach Induktionsvoraussetzung, und dies ist auch Modell von C nach Lemma 7.14.

Die Tatsache, daß jedes solche C ein Modell hat, ist aber widersprüchlich zu $S \vdash_R \Box$. ∎

Satz 7.18 *Sei S eine PL-Klauselmenge. Wenn S unerfüllbar ist, dann gilt* $S \vdash_R \Box$.

Beweis: Nach dem Satz von Herbrand gibt es eine endliche Menge $S' \subset Term_\Sigma^0(S)$ von Grundinstanzen von S, die unerfüllbar ist.

Diese ist AL-unerfüllbar über der Signatur At_Σ^0.

Mit der Vollständigkeit der AL-Resolution folgt, daß $S' \vdash_{RA} \Box$ gilt mit einer Ableitung $\Gamma' = \langle C_1', \ldots, C_n' \rangle$.

Aus Γ' läßt sich via Lifting eine PL-Ableitung Γ von $\Box$ aus S konstruieren, so daß für eine geeignete Grundsubstitution η gilt: $\Gamma\eta = \Gamma'$. ∎

Wie bei Klauseltableaus kann also der Vollständigkeitsbeweis für Resolution in die Teile Grundvollständigkeit und Lifting aufgeteilt werden. Die Grundvollständigkeit wird im folgenden in Satz 7.26, die Lifting Eigenschaft wird in Lemma 7.20 bewiesen.

7.2 Lifting

Es gibt zwei Probleme:

1. In der PL-Ableitung sind zusätzliche Faktorisierungsschritte notwendig.

2. Die Substitutionen in den PL-Resolutionsschritten müssen mgus sein.[2]

[2] Beim Lifting von Tableaus waren dagegen beliebige Grundsubstitutionen ausreichend. Informell gesprochen ist der Grund dafür, daß ein Tableau einen einzigen Beweis darstellt, während eine Menge von Resolventen implizit viele Ableitungen enthält. Daher kann zum Abschluß aller Tableauzweige dieselbe Substitution verwendet werden deren Anwendung erst ganz zum Schluß zu erfolgen braucht, während verschiedene Resolutionsableitungen verschiedene Substitutionen erfordern. Dabei muß die allgemeinste mögliche (eben der mgu) gleich angewandt werden. Dafür brauchen mgus in Resolutionsschritten aber auch nur lokal angewandt werden. Der Preis für diese Lokalität drückt sich in einem aufwendigeren Lifting-Lemma aus.

Notation: mit $\mathcal{R}^n(S)$ bezeichnen wir sämtliche Klauseln (modulo gebundener Umbenennung), die mit einer Resolutionsableitung der höchstens der Länge n aus S gewonnen werden können. Falls S endlich ist, dann ist auch $\mathcal{R}^n(S)$ endlich (Beweis: Übung).

Der Beweis des Lifting-Lemmas ist wegen seines technischen Charakters etwas mühsam einzusehen. Deswegen wollen wir ihn mit einem ausführlichen Beispiel motivieren:

Beispiel 7.19 Gegeben sind Klauseln $C = \{p(x), p(f(y))\}$, $D = \{\neg p(x), q(x)\}$. Weiterhin sind $\rho = \{x \leftarrow f(a), y \leftarrow a\}$ und $\tau = \{x \leftarrow f(a)\}$ Grundsubstitutionen von C bzw. D. Somit gilt $C\rho = \{p(f(a))\}$, $D\tau = \{\neg p(f(a)), q(f(a))\}$, was man in AL zu $E = \{q(f(a))\}$ resolvieren kann.

Im folgenden unterscheiden wir nicht immer zwischen $\{L\}$ und L, um die Notation zu vereinfachen.

Offensichtlich gilt $E = (C - L)\rho \cup (D - M)\tau$ mit $L = C$, $M = \{\neg p(x)\}$, $|L\rho| = |M\tau| = 1$ und $L\rho = \overline{M\tau}$.

Wir müssen einen PL-Resolutionsschritt (mit Faktorisierung) aus C und D konstruieren, dessen Resolvente E als Grundinstanz hat. Dies geschieht wie folgt:

1. Variablenumbenennung, damit C und D variablendisjunkt werden: wir verwenden $C\xi_C$, $D\eta_D$ mit $\xi_C = \{\}$ und $\eta_D = \{x \leftarrow u\}$.

2. Gewinne aus ρ und τ einen Unifikator θ von $\{L\xi_C, \overline{M\eta_D}\}$:
$$\theta = \{x \leftarrow f(a),\ y \leftarrow a,\ u \leftarrow f(a)\}$$
Damit gilt $E = (C - L)\xi_C\theta \cup (D - M)\eta_D\theta$.

3. Führe geeignete Faktorisierung von $C\xi_C$, $D\eta_D$ durch mittels $\nu_C = \{x \leftarrow f(y)\}$ und $\mu_D = \{\}$.

4. Berechne den mgu σ_N der umbenannten, faktorisierten Klauseln, d.h. von
$$N = \{L\xi_C\nu_C, \overline{M\eta_D\mu_D}\} = \{p(f(y)), \neg p(u)\}\ .$$
Dies ergibt $\sigma_N = \{u \leftarrow f(y)\}$.

5. Da σ_N ein mgu ist, kann man es zu θ erweitern mit einer geeigneten Substitution $\lambda = \{y \leftarrow a\}$. Also gilt $\theta = (\nu_C \cup \mu_D)\sigma_N\lambda$.

Insgesamt ist nun ein geeigneter PL-Resolutionsschritt gefunden, wobei $C\xi_C$ und $D\eta_D$ Umbenennungen von C und D sind, $C\xi_C\nu_C$, $D\eta_D\mu_D$ durch Faktorisierung entstanden sind und
$$E' = ((C - L)\xi_C\nu_C \cup (D - M)\eta_D\mu_D)\sigma_N = \{q(f(y))\}$$

eine PL-Resolvente von $C\xi_C\nu_C$, $D\eta_D\mu_D$ ist. E ist eine Grundinstanz von E', da $E = E'\lambda$. Es reicht aus, dieses Beispiel im allgemeinen Fall nachzuvollziehen. ∎

Wie schon angedeutet, ist die entscheidende Eigenschaft der Resolutionsregel auf das Lifting bezogen die folgende:

Lemma 7.20 *Für beliebige prädikatenlogische* Σ*-Klauselmengen* S *und* $T \subseteq$ *Term*$^0_\Sigma$ *gilt:* $\mathcal{R}(T(S)) \subseteq T(\mathcal{R}(S))$.

Beweis: Angepaßt von [Robinson, 1965b, S. 408f]. Der Beweis des benutzten Unifikationstheorems sowie der benötigten Eigenschaften von Substitutionen ist ebendort oder in Schmitt [1992a] zu finden.

Sei $E \in \mathcal{R}(T(S))$ und nicht schon in $T(S)$, sonst ist die Aussage trivial. Also ist E AL-Resolvente zweier Grundklauseln $C\rho$ und $D\tau$ aus $T(S)$, wobei $C, D \in S$. Es ist also $E = (C - L)\rho \cup (D - M)\tau$ für nichtleere $L \subseteq C$, $M \subseteq D$. Außerdem gilt $\overline{L\rho} = M\tau$ und $|L\rho| = |M\tau| = 1$.

Es sei ρ durch $\{v_1 \leftarrow t_1, \ldots, v_k \leftarrow t_k\}$ und τ durch $\{u_1 \leftarrow w_1, \ldots, u_m \leftarrow w_m\}$ gegeben, wobei $\mathrm{Var}(C) = \{v_1, \ldots, v_k\}$ und $\mathrm{Var}(D) = \{u_1, \ldots, u_m\}$.

Seien ξ_C und η_D Variablenumbennenungen von C und D mit frischen Variablen $\{x_1, \ldots, x_k\}$ und $\{y_1, \ldots, y_m\}$. Wir betrachten die Vereinigung θ von ρ und τ auf diesen Variablen:

$$\theta = \{x_1 \leftarrow t_1, \ldots, x_k \leftarrow t_k, y_1 \leftarrow w_1, \ldots, y_m \leftarrow w_m\}$$

Damit ist $E = (C-L)\xi_C\theta \cup (D-M)\eta_D\theta$, sowie $L\xi_C\theta = L\rho$ und $M\eta_D\theta = M\tau$. Insbesondere wird also $L\xi_C \cup \overline{M\eta_D}$ von θ unifiziert. Dann gibt es auch mgus ν_C von $L\xi_C$ und μ_D von $\overline{M\eta_D}$, sowie den mgu σ_N von $N = L\xi_C\nu_C \cup \overline{M\eta_D\mu_D}$. Außerdem existiert eine Substitution $\lambda : \mathrm{Var} \to T$, so daß $\theta = (\nu_C \cup \mu_D)\sigma_N\lambda$.

Insbesondere gilt $|L\xi_C\nu_C| = |M\eta_D\mu_D| = 1$ und $L\xi_C\nu_C\sigma_N\lambda = L\rho$ sowie $M\eta_D\mu_D\sigma_N\lambda = M\tau$. Daraus folgt, daß $\overline{L\xi_C\nu_C\sigma_N} = M\eta_D\mu_D\sigma_N$. Wir fassen zusammen:

- $C' = C\xi_C$ und $D' = D\eta_D$ sind durch gebundene Umbenennung aus C und D entstanden;
- $\mathrm{Var}(C') \cap \mathrm{Var}(D') = \emptyset$;
- $C'\nu_C$ und $D'\mu_D$ sind durch Faktorisierung aus C' und D' entstanden;
- σ_N ist mgu der einzigen Literale in $L' \in L\xi_C\nu_C$ und $M' \in M\eta_D\mu_D$.
- Also ist die Klausel

  $$E' = ((C'\nu_C - L') \cup (D'\mu_D - M'))\sigma_N$$

  eine Resolvente mit Faktorisierung von C' und D'.

Damit gilt, daß $E' \in \mathcal{R}(S)$, und weiterhin:

$$\begin{aligned} E &= (C-L)\rho \cup (D-M)\tau \\ &= (C-L)\xi_C\theta \cup (D-M)\eta_D\theta \\ &= ((C'-L\xi_C)(\nu_C \cup \mu_D) \cup (D'-M\eta_D)(\nu_C \cup \mu_D))\sigma_N\lambda \\ &= ((C'\nu_C - L') \cup (D'\mu_D - M'))\sigma_N\lambda \\ &= E'\lambda \in T(\mathcal{R}(S)) \end{aligned}$$

■

Aus Lemma 7.20 folgt durch eine einfache Induktion das

Korollar 7.21 *Für beliebige PL-Σ-Klauselmengen* S, $T \subseteq \mathit{Term}^0_\Sigma$ *und* $n \in \mathbb{N}$ *gilt:* $\mathcal{R}^n(T(S)) \subseteq T(\mathcal{R}^n(S))$.

Dies ist gerade die im Beweis von Satz 7.18 benötigte Eigenschaft.

Lemma 7.20 läßt sich noch in zweierlei Hinsicht verallgemeinern: zum einen kann man Klauseln als *Folgen* von Literalen auffassen. Damit gilt zum Beispiel $\{q, q\} \neq \{q\}$. Das Lemma gilt dann noch genauso, allerdings wird das Lifting der Faktorisierung technisch etwas aufwendiger. Wir werden dieses verfeinerte Klauselkonzept später benötigen, verzichten jedoch auf den Beweis des Lifting-Lemmas. Außerdem gilt das Lifting-Lemma auch wenn T eine beliebige Menge von Termen ist, nicht nur für Grundterme. Einen Beweis, der beide Verallgemeinerungen berücksichtigt, kann man im Buch von Leitsch (1997) finden.

7.3 Grundvollständigkeit

Es bleibt, die Vollständigkeit der AL-Resolution zu zeigen.

Ich demonstriere zunächst eine klassische Beweistechnik, die auch sonst eine Rolle im Automatischen Beweisen spielt, nämlich semantische Bäume, die von Robinson [1968] und Kowalski and Hayes [1969] entwickelt wurden.

Definition 7.22 *Für eine beliebige Aufzählung* $\{p_0, p_1, \ldots\}$ *einer AL-Signatur* Σ *ist der* semantische Baum ST_Σ *als ein markierter, binärer Baum mit folgenden Eigenschaften definiert:*

1. *Die Wurzel ist unmarkiert.*
2. *Die beiden Nachfolgerknoten der Wurzel sind mit* p_0 *bzw.* $\neg p_0$ *markiert.*
3. *Für jeden mit* p_i *oder mit* $\neg p_i$ *markierten Knoten sind die beiden Nachfolgerknoten mit* p_{i+1} *bzw.* $\neg p_{i+1}$ *markiert.*

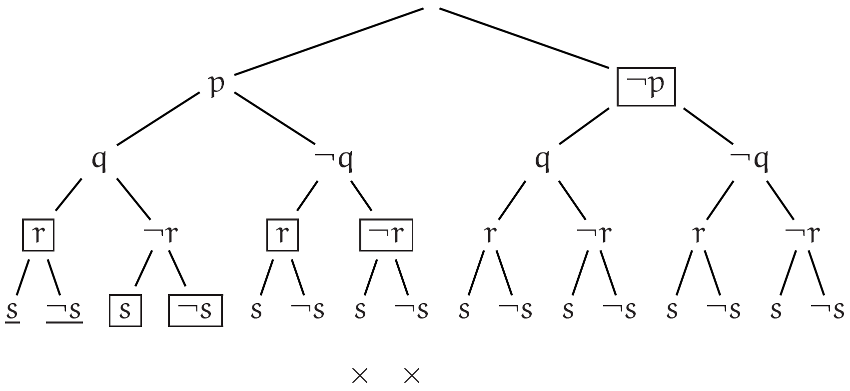


Abb. 7.3: Semantischer Baum für $\Sigma_S = \{p, q, r, s\}$.

■

Semantische Bäume repräsentieren alle möglichen Σ-Interpretationen.

Definition 7.23 *Sei* ST_Σ *ein semantischer Baum für* Σ. *Die* Widerlegungsmenge *eines Knotens* v *in* ST_Σ *besteht aus sämtlichen Markierungen oberhalb und einschließlich der von* v.

Eine AL-Klausel C schlägt fehl *bei* v, *falls für alle Literale* $L \in C$ *die Markierung* $\overline{L}$ *in der Widerlegungsmenge von* v *auftaucht. Ein Knoten* v *ist ein* Fehlschlagsknoten *für eine AL-Klauselmenge* S, *wenn eine Klausel* $C \in S$ *bei* v *fehlschlägt und keine Klausel aus* S *oberhalb von* v *fehlschlägt.*

Ein Knoten, dessen beide Nachfolger Fehlschlagsknoten sind, heißt Inferenzknoten.

ST_Σ *heißt* geschlossen *für* S, *wenn jeder Pfad einen Fehlschlagsknoten für* S *enthält.* ■

Beispiel 7.24 $S = \{\{p\}, \{r, \neg p, q\}, \{s, \neg q\}, \{\neg q, \neg s\}, \{\neg q, \neg r\}, \{\neg r, q\}\}$. Die Signatur von S ist $\Sigma_S = \{p, q, r, s\}$. In Abbildung 7.3 ist der semantische Baum zu Σ_S dargestellt. Fehlschlagsknoten von S sind eingerahmt. Würde die Klausel $\{r, \neg p, q\}$ aus S entfernt, so wären die mit $\times$ markierten Pfade offen, da sie keinen Fehlschlagsknoten mehr enthielten. Man beachte, daß die zwei unterstrichenen Knoten keine Fehlschlagsknoten sind: zwar schlagen die Klauseln $\{s, \neg q\}$ bzw. $\{\neg q, \neg s\}$ bei ihnen fehl, darüber jedoch schlägt schon $\{\neg q, \neg r\}$ fehl.
■

Die Pfade, die keinen Fehlschlagsknoten für S enthalten, repräsentieren gerade die Modelle von S, insbesondere gilt:

Lemma 7.25 *Ist eine endliche AL-Klauselmenge* S *über der Signatur* Σ *unerfüllbar, so ist jeder semantische Baum über* Σ *geschlossen für* S.

Beweis: Wir betrachten einen beliebigen Pfad π in einem beliebigen semantischen Baum über Σ ohne Fehlschlagsknoten für S. Jedes Element aus Σ taucht nach Definition semantischer Bäume genau einmal entweder positiv oder negativ unter den Markierungen von π auf.

Die Markierungen $M = \{L_0, L_1, \ldots\}$ auf π definieren also eine Σ-Interpretation durch

$$I(p) = \begin{cases} W & \text{falls } p \in M \\ F & \text{sonst} \end{cases}$$

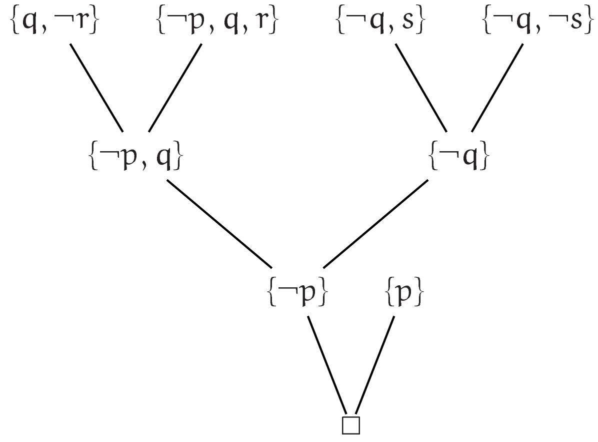
Da S unerfüllbar ist, kann I kein Modell von S sein, also muß $\mathrm{val}_I(C) = F$ für ein $C \in S$ gelten. Dafür muß aber nach Definition von I das Komplement eines jeden Literals von C in M auftauchen. Es muß also doch einen Fehlschlagsknoten auf π geben – Widerspruch. ■

Satz 7.26 *Sei S eine unerfüllbare AL-Klauselmenge. Dann gilt* $S \vdash_{RA} \Box$.

Beweis: Sei ST ein geschlossener semantischer Baum für S nach Lemma 7.25. Sei v ein Inferenzknoten maximaler Tiefe in ST. Die Nachfolger von v sind Fehlschlagsknoten. Diese seien mit p und $\neg p$ markiert. Seien $C, D \in S$ Klauseln, die bei den Nachfolgern von v fehlschlagen. Also muß $C = \{\neg p\} \cup C'$ und $D = \{p\} \cup D'$ für geeignete C', D' gelten. C und D können somit AL zu $E = C' \cup D'$ resolviert werden. E schlägt bei v fehl, da die Komplemente aller Literale in $C \cup D$ außer p und $\neg p$ schon in der Widerlegungsmenge von v auftauchen. Die Klauselmenge $S \cup \{E\}$ hat also einen geschlossenen semantischen Baum mit weniger Fehlschlagsknoten als ST.

Da ein geschlossener semantischer Baum nur endlich viele Fehlschlagsknoten hat, gibt es eine AL-Resolutionsableitung $\langle C_1, \ldots, C_n \rangle$ aus S, so daß $\overline{S} = S \cup \bigcup_{i=1}^{n} \{C_i\}$ einen geschlossenen semantischen Baum besitzt, dessen einziger Fehlschlagsknoten die Wurzel ist. Die einzige Klausel, die für diesen semantischen Baum fehlschlagen kann, ist aber $\Box$, weshalb $\Box \in \overline{S}$. ■

Beispiel 7.27 Der Beweis von Satz 7.26 gibt in konstruktiver Weise an, wie man aus einem geschlossenen semantischen Baum zu einer Formelmenge S eine Resolutionsableitung von $\Box$ aus S konstruieren kann. Wenn wir das für S und den semantischen Baum aus Beispiel 7.24 tun, ergibt sich folgende Ableitung (in Baumdarstellung):

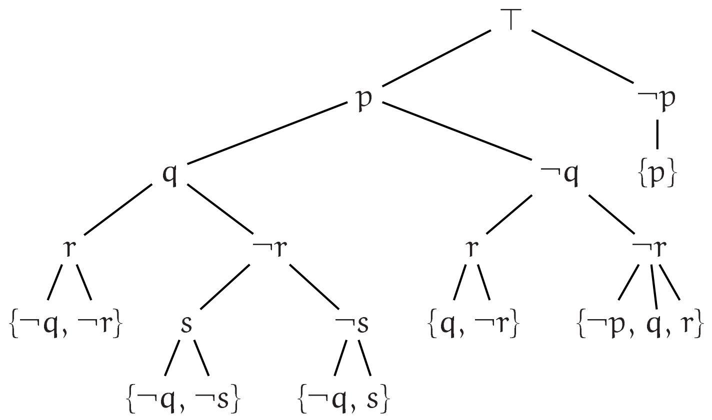


■

Varianten von semantischen Bäumen kann man auch dazu verwenden, die Vollständigkeit von Tableauprozeduren zu beweisen (siehe Klingenbeck and Hähnle [1994]).

Man kann semantische Bäume als reguläre Klauseltableaus mit atomaren Schnitten auffassen, wobei die Anwendung der üblichen Erweiterungsregel nur erlaubt ist, sofern alle neuen Zweige sofort abgeschlossen werden können (damit wird erreicht, daß Zweige nur direkt unterhalb von Fehlschlagsknoten abgeschlossen werden).

Wird der semantische Baum aus Abbildung 7.3 unterhalb der Fehlschlagsknoten abgeschnitten, so kann der entstehende Unterbaum leicht zu einem Klauseltableau (kein Konnektionstableau) mit atomarem Schnitt erweitert werden:



Semantische Bäume sind ebenfalls eng verwandt mit der sogenannten Davis-Putnam Prozedur: Davis and Putnam [1960]; Davis *et al.* [1962].

Die Resolutionsableitung im Beweis von Satz 7.26 ist i.a. nicht die Kürzestmögliche, da ihre Länge stark von der gewählten Ordnung abhängt.

Jede mögliche Ableitung unter einer fest vorgegebenen Ordnung auf dem Baum kann sogar exponentiell länger sein als eine unbeschränkte Resolutionsableitung (das folgt aus der Arbeit von Goerdt [1993]).

Verwendete/Empfohlene Literatur zu den vorangegangenen und kommenden Abschnitten Das Originalpapier von Robinson [1965b] ist klar geschrieben und sehr empfehlenswert.

Eine Standardreferenz für Resolution, die den Stand bis 1973 zusammenfasste, war das Buch von Chang and Lee [1973]. Obwohl manchmal etwas umständlich, enthielt es viele Beispiele und war ganz gut lesbar (leider enthielt es auch einige Fehler). Demgegenüber bringen Wos *et al.* [1984] nichts wesentliches. Beide Werke sind inzwischen jedoch überholt.

Die derzeit umfassendste und modernste Darstellung der Resolutionstheorie ist die von Leitsch [1997]. Sehr zu empfehlen ist auch der Artikel von Bachmair and Ganzinger [1997], der eine uniforme Darstellung der meisten Resolutionsvarianten versucht.

Die wichtigsten Originalartikel sind in der Sammlung von Siekmann and Wrightson [1983b,c] enthalten.

Semantische Bäume wurden von Robinson [1968] eingeführt und von Kowalski and Hayes [1969] verallgemeinert.

7.4 Implementierung von Resolution

Im folgenden verwenden wir die Abkürzung $\mathcal{R}^*(S)$ für $\bigcup_{i=1}^{\infty} \mathcal{R}^i(S)$.

Für jede endliche Klauselmenge S muß von einer vollständigen Implementierung jede mögliche Resolvente, also jedes Element aus $\mathcal{R}^*(S)$ (modulo gebundener Umbenennung), in endlich vielen Schritten erzeugt werden.

Dies kann einfach durch *stufenweise Saturierung* (level saturation) geschehen:

```
/* Init */

S := [];
while ¬eof do S := append(S,[read_next_clause]);

/* Saturate */

while □ ∉ S do S := R(S);
```

Nachteile dieses Vorgehens:

1. Zu grobkörnig; Verfeinerungen beziehen sich meist auf einzelne Resolutionsschritte.

2. Größe von S (Anzahl der Literale) wächst von einer Stufe zur nächsten zu schnell an (kubisch, da die Anzahl der neuen Resolventen quadratisch in $|S|$ ist und deren Länge linear in $|S|$).

Stattdessen werden die neuen Resolventen einer einzelnen Klausel mit den übrigen Klauseln berechnet.

```
while □ ∉ S do begin
  C := first(S);
  T := {E | E Resolvente von C,D ∈ S, E ∉ S};
  S := append(rest(S),append(T,[C]))
end;
```

Auch diese Version hat noch einen entscheidenden Nachteil: eine Resolvente aus denselben Elternklauseln kann mehrfach erzeugt werden (zum Beispiel zuerst aus `C` und `D`, wenn `first(S)=C`, dann aus `D` und `C`, wenn `first(S)=D`), wenngleich sie nur einmal behalten wird. Dieses Problem wird später in Zusammenhang mit der sogenannten Stützmengenstrategie behoben werden.

Die Implementierung über eine Schlange ähnelt der fairen Implementierung von `select_clause` bei Klauseltableaus. Dadurch wird jede Klausel in $\mathcal{R}^*(S)$ nach endlicher Zeit generiert, das heißt, Klauseln werden in *fairer Weise* zu Resolutionsschritten herangezogen. Die Prozedur ist allerdings deutlich einfacher als die Klauseltableausprozedur, da es ausreicht alle Resolventen zu bilden. Man sagt daher von Resolution auch, daß sie ein auf *Saturierung* einer Klauselmenge beruhendes Verfahren sei. Bei Konnektionstableaus und bei Klauseltableaus mit Backtracking müssen alle möglichen *Beweise* generiert werden. Außerdem ist die vorgestellte Resolutionsprozedur offenbar *beweiskonfluent*.

Kapitel 8

Varianten/Verfeinerungen des Resolutionskalküls

Übersicht und Einteilung verschiedener Verfeinerungen von Resolution
Ziel ist es jeweils, weniger Resolventen erzeugen zu müssen, um einen Beweis zu konstruieren. Dies kann auf verschiedene Weise erreicht werden: man kann bestimmte Resolventen (geordnete, Lock-Resolution, teils semantische Resolution) oder Elternklauseln (pure Literale, Tautologien) verbieten, man kann auch bestimmte Resolutionsableitungen (lineare, Input, Unit Resolution) ausschließen:

1. Löschen von Klauseln, die nicht zur Ableitung von $\Box$ beitragen.
   (a) Pure Literale
   (b) Tautologien
   (c) Subsumtion
2. Einschränkung der Möglichkeiten zur Bildung neuer Resolventen.
   (a) geordnete Resolution
   (b) Lock-Resolution
3. Einschränkung der Form der Resolutionsableitung.
   (a) Lineare Resolution
   (b) Input Resolution
   (c) Unit Resolution
4. Kombination von 2. und 3.: semantische Resolution.
   (a) Hyperresolution

(b) Stützmengenstrategie (Set of support – SOS)

(c) UR-Resolution

(d) Geordnete Resolution mit Selektionsfunktion

8.1 Löschstrategien

Definition 8.1 *Sei* S *eine PL-Klauselmenge.* $C \in S$ *heißt* pur *in* S*, falls es ein* $L \in C$ *gibt, so daß* L *kein Link mit irgendeiner Klausel aus* S *bildet.* $C \in S$ *heißt* tautologisch *falls* C *eine AL-Tautologie ist.* C subsumiert D *falls es eine Substitution* σ *gibt, so daß* $C\sigma \subseteq D$*.* ■

Beispiel 8.2 S enthalte die folgenden Klauseln

$$\begin{array}{lll} C_1 = \{p(x,z), q(z)\} & C_2 = \{q(a), p(b,a), r(u)\} & C_3 = \{p(x,y), \neg r(f(x))\} \\ C_4 = \{\neg p(y,y), \neg q(y)\} & C_5 = \{p(z,x), r(g(z))\} & C_6 = \{\neg r(f(x)), \neg q(z)\} \end{array}$$

C_5 ist pur in S; C_2 wird von C_1 subsumiert; C_1 und C_4 haben die PL-Resolvente $\mathrm{RF}(C_1, 1; C_4, 1) = \{q(y), \neg q(y)\}$, die tautologisch ist. Dagegen ist die zweite Resolvente $\mathrm{RF}(C_1, 2; C_4, 2) = \{p(x,y), \neg p(y,y)\}$ nicht tautologisch, da x und y verschieden instanziiert werden können. ■

Wir führen vier Löschoperatoren D_x, wobei $x \in \{\mathrm{pur}, \mathrm{taut}, \mathrm{sub}, \overline{\mathrm{sub}}\}$, auf Klauselmengen ein:

$$\begin{array}{ll} D_{\mathrm{pur}}(S) & = S - \{C \in S \mid C \text{ pur in } S\}. \\ D_{\mathrm{taut}}(S) & = S - \{C \in S \mid C \text{ tautologisch}\}. \\ D_{\mathrm{sub}}(S,T) & = S - \{D \in S \mid \exists C \in T : \ C \text{ subsumiert } D\}. \\ D_{\overline{\mathrm{sub}}}(S) & = S - \{D \in S \mid \exists C \in S : \ C \neq D, \ C \text{ subsumiert } D\}. \end{array}$$

Dabei ist $\neq$ in $D_{\overline{\mathrm{sub}}}$ modulo gebundener Umbenennung zu verstehen, sonst würde stets $D_{\overline{\mathrm{sub}}}(S) = \emptyset$ gelten.

Alle Löschoperatoren erhalten die Modelle einer Klauselmenge, da keine neuen Klauseln hinzukommen.

8.1.1 Implementierung der Löschregeln

1. Einmalig als Vorverarbeitungsschritt.

2. Immer nach der Berechnung neuer Resolventen wie folgt:

```
while □ ∉ S do begin
  C := first(S);
  T := {E | E Resolvente von C,D ∈ S, E ∉ S};
  Ξ;
  S := append(rest(S),append(T,[C]))
end;
```

Dabei ist für Ξ möglich:

$T = D_{pur}(T)$
$T = D_{taut}(T)$
$T = D_{sub}(T,S)$ (*Vorwärtssubsumtion*)
$S = D_{sub}(S,T)$ (*Rückwärtssubsumtion*)

oder Kombinationen hiervon.

Beispiel 8.3 Sei S wie im vorigen Beispiel definiert. Die prädikatenlogische Resolvente mit Faktorisierung $RF(C_3, 1; C_4, 1) = \{\neg r(f(y)), \neg q(y)\}$ wird von C_6 subsumiert und bei Vorwärtssubsumtion gelöscht. ■

Es ist auch möglich, die letzten beiden Zeilen im Rumpf der *while* Schleife durch folgendes Programmfragment zu ersetzen:

```
S := D_{\overline{sub}}(S∪T)∩S;
T := D_{\overline{sub}}(S∪T)∩T;
if C∈S then
  S := append(rest(S),append(T,[C]))
else
  S := append(S,T);
```

Dies nennt man *totale Subsumtion*. Dabei werden S und T getrennt berechnet, damit abhängig davon ob C noch in T enthalten ist, T oder [C|T] an S gehängt werden kann.

Satz 8.4 *Die Subsumtionsrelation auf PL-Klauseln ist entscheidbar, genauer: sie ist NP-vollständig.*

Beweis: Siehe Garey and Johnson [1979]. ■

Wir wollen einen konkreten Subsumtionsalgorithmus vorstellen. Subsumtion kann mit eingeschränkter Resolution entschieden werden. Zunächst betrachtet man ein spezielleres Problem:

Definition 8.5 *Ein Literal* L *ist eine (σ-)*Instanz *eines Literals* K*, falls es eine Substitution* σ *gibt, so daß* $L\sigma = K$. ■

Proposition 8.6 L *ist eine Instanz von* K *gdw* L *und* $K\theta$ *unifizierbar sind, wobei* $\theta = \{x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n\}$*,* $Var(K) = \{x_1, \ldots, x_n\}$ *und* $\{a_1, \ldots, a_n\}$ *sind neue Konstanten, die in* K, L *nicht vorkommen.*

Beweis: Übung. ∎

Ein Subsumtionsalgorithmus kann nun wie folgt hergeleitet werden:
Seien C, $D = \{K_1, \dots, K_m\}$ PL-Klauseln.

C subsumiert D
gdw
es eine Substitution σ gibt, so daß $C\sigma \subseteq D$
gdw
es eine Substitution σ gibt, so daß jedes $L \in C$ σ-Instanz eines $K \in D$ ist
gdw (Proposition 8.6)
jedes $L \in C$ mit einem $K_i\theta_i$ simultan unifizierbar ist, wobei die in den θ_i eingeführten Konstanten disjunkt sind
gdw (Übung)
$\Box \in \mathcal{R}^*(\{\{\neg K_1\theta_1\}, \dots, \{\neg K_m\theta_m\}, C\})$, wobei keine Resolutionsschritte innerhalb von C oder zwischen den $\neg K_i\theta_i$ erlaubt sind.

Beispiel 8.7 Sei $C = \{p(x, y), q(z)\}$ und $D = \{q(a), p(b, z), r(u)\}$. Es ist also

$$\Box \in \mathcal{R}^*(\{\{\neg q(a)\}, \{\neg p(b, c)\}, \{\neg r(d)\}, C\})$$

zu testen: C und $\{\neg q(a)\}$ resolvieren zu $\{p(x, y)\}$, welches mit $\{\neg p(b, c)\}$ zu $\Box$ resolviert. Daher wird D von C subsumiert.

Ersetzt man aber C durch $C' = \{p(x, y), q(x)\}$, ergibt die Resolution von C' mit $\{\neg q(a)\}$ die Resolvente $\{p(a, y)\}$, auf die nun kein weiterer Resolutionsschritt anwendbar ist. Wenn man mit C' und $\{\neg p(b, c)\}$ anfängt geht es ähnlich, so daß also C' nicht D subsumiert. ∎

Nachteil: Es werden im schlimmsten Fall überexponentiell viele Resolventen erzeugt, was unnötig ist, da Subsumtion ja in NP liegt. Bessere Algorithmen mit exponentieller Laufzeit im schlechtesten Fall wurden von Gottlob and Leitsch [1985] angegeben.

8.1.2 Vollständigkeit der Löschregeln

Eine einfache Möglichkeit, die Vollständigkeit der Löschregeln zu beweisen, bieten semantische Bäume für PL. Diese sind genau wie semantische Bäume in der AL definiert, aber über einer Aufzählung der Grundatome der Signatur Σ einer gegebenen PL-Klauselmenge S.

Definition 8.8 *PL-Klausel* C schlägt fehl *bei* v *falls es eine Grundsubstitution* σ *gibt, so daß für alle Literale* $L \in C\sigma$ *die Markierung* $\overline{L}$ *in der Widerlegungsmenge von* v *auftaucht.* ∎

Alle übrigen Definition bleiben genau gleich.

Lemma 8.9 *Ist die PL-Klauselmenge S über Σ unerfüllbar, dann gibt es eine endliche Menge $K \subset At^0_\Sigma$, so daß jeder semantische Baum über K geschlossen ist für S.*

Beweis: Fast genau wie im AL-Fall zeigt man zunächst, daß jeder semantische Baum ST über At^0_Σ geschlossen ist für S (Übung).

In einem solchen ist aber die Anzahl der Knoten, die oberhalb irgendeines Fehlschlagsknotens liegen, endlich, denn sonst könnte man nach Königs Lemma einen unendlichen Zweig ohne Fehlschlagsknoten finden.

Folgendes ist also wohldefiniert: $K = \{p_1 \ldots, p_k\}$, wobei k die maximale Tiefe eines Fehlschlagsknotens in ST ist, und die Aussage ist für K offenbar erfüllt. ∎

Man beachte, daß die Anordnung der Elemente von At^0_Σ keine Rolle für den Beweis spielt.

Lemma 8.9 ist im wesentlichen eine Variante des Satzes von Herbrand.

Wir geben nun einen alternativen Beweis von Satz 7.18, der nicht den Umweg über die Grundvollständigkeit geht.

Satz 8.10 *Sei S eine unerfüllbare PL-Klauselmenge. Dann gilt $S \vdash_R \square$.*

Beweis: Sei ST ein geschlossener semantischer Baum für S nach Lemma 8.9. Sei v ein Inferenzknoten mit maximaler Tiefe in ST. Die Nachfolger von v sind Fehlschlagsknoten. Diese seien mit p und $\neg p$ markiert. Seien $C, D \in S$ Klauseln, die bei den Nachfolgern von v fehlschlagen. Also muß $C\sigma = \{\neg p\} \cup C'$ und $D\tau = \{p\} \cup D'$ für geeignete C', σ, D', τ gelten. Somit können $C\sigma$ und $D\tau$ AL zu $E' = C' \cup D'$ resolviert werden.

Das Lifting-Lemma garantiert, daß C und D zu E PL-resolviert werden können, wobei E' eine Instanz von E ist.

E' schlägt bei v fehl, da die Komplemente aller Literale in $C' \cup D'$ außer p und $\neg p$ schon in der Widerlegungsmenge von v auftauchen.

Da E' eine Instanz von E ist, schlägt nach Definition des PL-Fehlschlags auch E bei v fehl. Die Klauselmenge $S \cup \{E\}$ hat also einen geschlossenen semantischen Baum mit weniger Fehlschlagsknoten als ST. Der restliche Beweis erfolgt wie in Satz 7.26. ∎

Die Vollständigkeit der Löschregeln kann man nun direkt am Beweis von Satz 8.10 ablesen:

Sei R eine Resolutionsableitung von $\square$ aus S, die aus obigem Beweis konstruiert wurde. Eine *pure Klausel* ist nicht in R, da ein pures Literal keinen Nachfolger eines Inferenzknotens markieren kann und also in keinem Resolutionsschritt auftaucht.

Eine *tautologische Klausel* C ist nicht in R, weil in keiner Widerlegungsmenge ein Literal *und* sein Komplement auftaucht. Letzteres ist aber für jede Instanz von C der Fall.

Seien C und D abgeleitete Klauseln vor einem beliebigen Resolutionsschritt in R. Es gelte $C\sigma \subseteq D$ und D schlage bei v fehl. Dann schlägt auch $D\tau$ für eine geeignete Grundsubstitution τ bei v fehl. Also schlägt erst recht $C\sigma\tau$ bei v fehl, mithin wird C und nicht D in R verwendet. Dies demonstriert die Vollständigkeit der *totalen Subsumtion*. Vorwärts- bzw. Rückwärtssubsumtion sind weniger einschränkend, also erst recht vollständig.

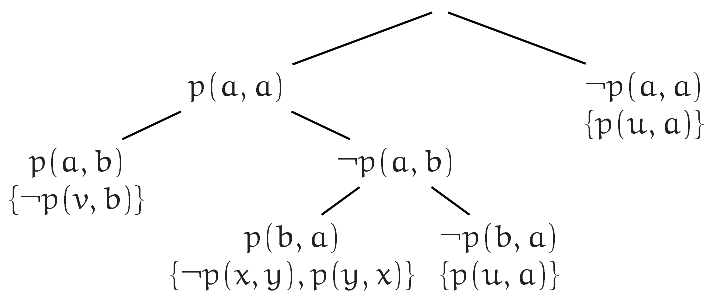
Beispiel 8.11 Sei

$$S = \{\{\neg p(x,y), p(y,x)\}, \{p(u,a)\}, \{\neg p(v,b)\}\} \ .$$

Ein geschlossener semantischer Baum existiert zum Beispiel über

$$K = \{p(a,a), p(a,b), p(b,a)\}$$

und sieht wie folgt aus (nur Knoten oberhalb und einschließlich der Fehlschlagsknoten sind dargestellt):



Die Substitution $\{x \leftarrow y\}$ macht $\{\neg p(x,y), p(y,x)\}$ zu einer tautologischen Klausel, die keinen Fehlschlagsknoten erzeugen kann. Selbstresolution mit dieser Klausel wird also durch die Tautologieregel ausgeschlossen. Da diese Klausel die Symmetrie des Prädikats p ausdrückt, ist dies ein wichtiger Fall.

Wie im Beweis von Satz 8.10 läßt sich eine Ableitung von $\square$ konstruieren: die Fehlschlagsknoten maximaler Tiefe sind $p(b,a)$ und $\neg p(b,a)$, mit widerlegenden Klauseln $\{\neg p(x,y), p(y,x)\}$ und $\{p(u,a)\}$, die zu $\{p(a,u)\}$ resolviert werden. In einem weiteren Schritt hat man daraus mit $\{\neg p(v,b)\}$ die leere Klausel.

Im allgemeinen ist ein geschlossener semantischer Baum nicht vorgegeben. Betrachtet man nur die Klauselmenge S, so ist auch ein initialer Resolutionsschritt zwischen $\{\neg p(x,y), p(y,x)\}$ und $\{\neg p(v,b)\}$ möglich. Beide Ableitungen sind sich sehr ähnlich. In Abschnitt 8.2.2 wird eine Einschränkung eingeführt, die einen von den beiden initialen Schritten ausschließt. ■

Historische Bemerkung: Die Löschung purer Literale sowie Subsumtion samt des angegebenen Algorithmus findet sich schon in Robinson [1965b]. Die Löschung von Tautologien ist eine Regel aus Davis and Putnam [1960]. Semantische Bäume für PL-Klauseln wurden in Kowalski and Hayes [1969] eingeführt.

8.2 Einschränkung der Resolventenbildung

8.2.1 Geordnete Resolution

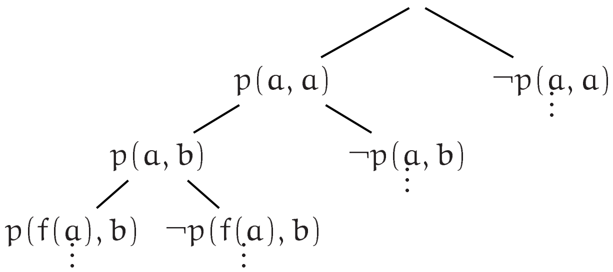
Semantische Bäume setzen eine beliebige (aber feste) Ordnung der Grundatome, die zu einer Klauselmenge gehören, voraus. Offenbar induziert diese eine $\mathcal{A}$-Ordnung, wobei $p < q$ gdw p im semantischen Baum näher bei der Wurzel ist als q.

Definition 8.12 *$<$ sei eine $\mathcal{A}$-Ordnung auf At_Σ. Ein semantischer Baum ST für At^0_Σ ist* kompatibel *mit $<$ falls für alle Markierungen K, L gilt: wenn $\mathsf{K} < \mathsf{L}$ dann ist der Knoten von K oberhalb des Knotens von L in ST.* ■

Zu jeder $\mathcal{A}$-Ordnung gibt es einen kompatiblen semantischen Baum: man ergänze $<$ zu einer totaler Ordnung (also: Aufzählung) auf At^0_Σ.

Die Resolutionsableitung, die aus dem Vollständigkeitsbeweis extrahiert werden kann, nimmt bezüglich dieser $\mathcal{A}$-Ordnung keine beliebigen Resolutionsschritte vor. Die im folgenden eingeführte Einschränkung macht sich dies zunutze, indem sie von vorne herein nur Resolutionsschritte zuläßt, die mit der durch einen semantischen Baum induzierten $\mathcal{A}$-Ordnung kompatibel sind.

Beispiel 8.13 Der folgende semantische Baum ist $\prec$-kompatibel (Def. 5.14) auf $\Sigma = \langle\{a, b, f\}, \{p\}, \{a/0, b/0, f/1, p/2\}\rangle$, aber er zählt nicht ganz At^0_Σ auf (dazu müsste man noch $p(b, a)$, $p(b, b)$ einfügen):



■

Für beliebige $\mathcal{A}$-Ordnungen $<$ ist offenbar jeder Knoten in einem $<$-kompatiblen semantischen Baum $<$-maximal in seiner Widerlegungsmenge.

Definition 8.14 *Sei* $<$ *eine* A*-Ordnung und* C*,* D *PL-Klauseln.* E *ist eine* geordnete Resolvente *von* C *und* D *bezüglich* $<$*, falls*

1. E *eine PL-Resolvente von* C *und* D *mit mgu* σ *ist,*
2. L *ist das resolvierte Literal, und*
3. $L\sigma$ *ist maximal in* E*.*

■

Die Faktorisierung wird nicht eingeschränkt, und geordnete Resolutionsableitungen sind analog zu gewöhnlichen Resolutionsableitungen definiert. A-Ordnungen und die geordnete Resolution sind als Spezialfall auch für AL erklärt.

Beispiel 8.15 $S = \{\{p(a)\}, \{\neg p(x), r(f(x))\}, \{\neg r(y), r(f(y))\}, \{\neg r(f^2(a))\}\}$. Aus den Klauseln $\{\neg r(f^2(a))\}$ und $\{\neg r(y), r(f(y))\}$ erhält man die $\prec$-geordnete Resolvente $\{\neg r(f(a))\}$, weil das resolvierte Literal $r(f^2(a))$ $\prec$-maximal in ihr ist. Daraus und aus $\{\neg p(x), r(f(x))\}$ erhält man die $\prec$-geordnete Resolvente $\{\neg p(a)\}$, in der $r(f(a))$ $\prec$-maximal ist. Schließlich wird damit und aus $\{p(a)\}$ die leere Klausel abgeleitet. Als erster Schritt hätte $\{p(a)\}$ nicht mit $\{\neg p(x), r(f(x))\}$ resolviert werden können, weil $p(a)$ nicht $\prec$-maximal in der Resolvente $\{r(f(a))\}$ ist.

Die Ordnung $\prec$ erzwingt in manchen Situationen die Erzeugung von flacheren Termen, allerdings nicht immer:

Aus $\{p(f(x)), q(x, f(x))\}$ und $\{\neg q(g(y), u)\}$ kann die $\prec$-geordnete Resolvente $\{p(f(g(y)))\}$ gewonnen werden, da $q(g(y), f(g(y)))$ $\prec$-maximal in ihr ist. ■

Satz 8.16 *Sei* S *eine unerfüllbare Klauselmenge und* $<$ *eine beliebige* A*-Ordnung. Dann gibt es eine* $<$*-geordnete Resolutionsableitung mit Faktorisierung von* $\square$ *aus* S*.*

Beweisskizze: Man muß sicherstellen, daß der Resolutionsschritt im Beweis von Satz 8.10 auch ein geordneter Resolutionsschritt ist.

Dazu wählt man nach Lemma 8.9 eine endliche Menge $K \subset \mathrm{Term}^0_\Sigma$, so daß jeder semantische Baum über K geschlossen ist für S. Da $<$ irreflexiv und transitiv ist und K endlich, kann man einen geschlossenen semantischen Baum ST, der kompatibel mit $<$ ist, wählen.

In ST gilt, daß die Markierung eines Fehlschlagknotens maximal in dessen Widerlegungsmenge ist (Irreflexivität von $<$ und Def.).

Wegen der Substitutivität (Verwendung der Kontraposition) gilt dies auch für das resolvierte Literal (also gerade diese Markierung) in der PL-Resolvente aus dem Lifting-Lemma. ■

Übung: führe den vorhergehenden Beweis detailliert aus.

Anwendungen von geordneter Resolution

1. Eine geringere Zahl möglicher Resolventen kann die Gesamtzahl der für einen Beweis zu bildenden Resolventen stark vermindern. Allerdings: der kürzeste Beweis mit geordneter Resolution ist i.a. exponentiell länger als der mit uneingeschränkter Resolution (das folgt aus der Arbeit von Goerdt [1993], da für eine geeignete Ordnung eine geordnete Resolutionsableitung auch eine reguläre Resolutionsableitung darstellt).

2. Die Menge aller geschlossenen PL-Formeln in PNF der Gestalt

   $$\{\exists x_1 \cdots \exists x_m \forall y \exists z_1 \cdots \exists z_n \phi \mid \phi \text{ quantorenfrei und ohne Funktionssymbole}\}$$

   ist entscheidbar hinsichtlich der Erfüllbarkeit ihrer Elemente; diese Formelklasse heißt in der Literatur *(erweiterte) Ackermann Klasse*.

   Die A-Ordnung $\prec$ kann verwendet werden, um dies zu zeigen (siehe Kapitel 5.2 von Leitsch [1997]). Hier eine Skizze der Vorgehensweise:

   Durch Skolemisierung und Berechnung der KNF der Matrix reduziert man die Formelklasse zunächst auf geschlossene Formeln der Gestalt

   $$\{\forall y \phi \mid \phi \text{ quantorenfrei und in KNF}\}.$$

   Man hat also Klauselmengen, in denen alle Klauseln C höchstens eine Variable enthalten, also $|\mathrm{Var}(C) \leq 1|$.[1] Man kann zeigen, daß für solche Klauseln die Termtiefe bei $\prec$-geordneten Resolutionsschritten nicht unbeschränkt zunehmen kann. Damit gibt es nur endlich viele Resolventen, und aus der Vollständigkeit der geordneten Resolution folgt, daß die Erfüllbarkeit für solche Formeln entschieden werden kann.

   Es gibt verschiedene weitere Formelklassen, die mit Variationen dieser Technik entschieden werden können, siehe Fermüller *et al.* [1993].

3. Für gewisse entscheidbare Formelklassen kann man Resolution damit als Entscheidungsverfahren und prinzipiell auch zur Gegenbeispielerzeugung verwenden (allerdings ist es nicht einfach aus einer saturierten Klauselmenge ein Modell zu konstruieren). Dies ist insbesondere bei Anwendungen in der Programmverifikation interessant (siehe Abschnitt 10.1).

8.2.2 Lock Resolution

Ähnlich wie geordnete Resolution benutzt auch Lockresolution eine Ordnung auf den Literalen in den Klauseln.

[1] Diese Klasse ist sogar allgemeiner als die Ackermannklasse. Deren Entscheidbarkeit wurde aber ursprünglich mit völlig anderen Mitteln gezeigt.

Definition 8.17 *Der Ausdruck* L^i *heißt* indiziertes Literal *wenn* L *ein Literal ist und* $i \in \mathbb{N}$. *Die Zahl* i *ist der* Index *von* L. *Eine* indizierte Klausel *ist eine endliche Menge indizierter Literale.* ■

Die Indizes in indizierten Klauselmengen können völlig beliebig sein; insbesondere können verschiedene Auftauchen desselben Literals verschiedene Indizes haben (verschiedene Literale können natürlich auch denselben Index haben, was dann eben weniger einschränkend ist).

Wie geordnete Resolution verkörpert Lock Resolution die Idee, daß Literale nicht in beliebiger Reihenfolge resolviert zu werden brauchen.

Definition 8.18 *Sei* $unlock(C) = \{L \mid L^i \in C\}$. *Eine indizierte Klausel* C *ist* erfüllbar *gdw* $unlock(C)$ *erfüllbar ist; entsprechendes gilt für Klauselmengen. Außerdem sei* $L^i\sigma = (L\sigma)^i$ *festgelegt.* ■

Identische Literale können verschiedene Indizes besitzen, weshalb eine explizite „Merging“ Operation benötigt wird. Falls ein Literal mit verschiedenen Indizes in einer Klausel auftaucht, wird nur das Auftauchen mit dem kleinsten Index behalten:

Definition 8.19 *Für indizierte Klauseln* C *ist die* reduzierte Klausel $r(C)$ *zu* C *definiert als* $r(C) = \{L^i \in C \mid$ *es gibt kein* $L^j \in C$ *mit* $j < i\}$. ■

Die Faktorisierung wird auf minimale Literale[2] eingeschränkt:

Definition 8.20 *Es sei* C *eine indizierte Klausel,* $L^i \in C$ *ein Literal mit minimalem Index und* σ *ein mgu von* $\{L^i, L_1^{i_1}, \ldots, L_m^{i_m}\} \subseteq C$*; dann heißt* $r(C\sigma)$ Lockfaktor *von* C. ■

Beispiel 8.21 $\{\overset{1}{p(x)}, \overset{2}{q(x)}, \overset{3}{q(a)}\}$ ist ein Lockfaktor von $\{\overset{1}{p(x)}, \overset{2}{p(y)}, \overset{2}{q(x)}, \overset{3}{q(a)}\}$, nicht aber $\{\overset{1}{p(x)}, \overset{2}{p(y)}, \overset{2}{q(a)}\}$. ■

Lock Resolution ist Resolution auf minimalen Literalen mit Lockfaktorisierung und anschließender Reduktion:

Definition 8.22 *Seien* C *und* D *indizierte Klauseln, die variablendisjunkt sind, und seien* C' *und* D' *Lockfaktoren von* C *und* D. *Weiterhin seien* $L^i \in C'$ *und* $M^j \in D'$ *Literale mit minimalem Index in* C' *bzw.* D', *und es gebe einen mgu* σ *von* L^i *und* $\overline{M}^j$. *Dann heißt*

$$r((C' - \{L^i\})\sigma \cup (D' - \{M^j\})\sigma)$$

Lockresolvente *von* C *und* D. ■

[2] Chang and Lee [1973] faktorisieren beliebige Literale. Da aber ohnehin immer nur über das minimale Literal resolviert wird, ist unsere Definition für die Vollständigkeit ausreichend.

Indizes von Literalen werden von den Elternklauseln auf die Resolvente vererbt. Bei doppeltem Auftauchen eines Literals in der Resolvente wird, entsprechend der Definition reduzierter Klauseln, der kleinere Index behalten.

Die Ableitungsrelation $\vdash_{RL}$ ist analog zu $\vdash_{RF}$ definiert.

Beispiel 8.23 Betrachten wir noch einmal die Klauselmenge aus Beispiel 8.11:

$$S = \{\{\neg p(\overset{1}{x}, y), p(\overset{2}{y}, x)\}, \{p(\overset{3}{u}, a)\}, \{\neg p(\overset{4}{v}, b)\}\}$$

Unter Absehung von den Indizes waren zwei Resolutionsschritte möglich, nämlich mit $\{\neg p(x, y), p(y, x)\}$ und $\{p(u, a)\}$, sowie mit $\{\neg p(x, y), p(y, x)\}$ und $\{\neg p(v, b)\}$. Keine A-Ordnung $<$ kann einen der beiden Schritte verhindern, da wegen ihrer Unifizierbarkeit niemals $p(u, a) < p(a, u)$ oder $p(v, b) < p(b, v)$ gelten kann.

Mit Lock Resolution kann unter Verwendung obiger Indizierung die zweite Resolutionsmöglichkeit ausgeschlossen werden, da das Literal $p(y, x)$ nicht minimal in $\{\neg p(\overset{1}{x}, y), p(\overset{2}{y}, x)\}$ ist und somit nicht für den ersten Resolutionsschritt zur Verfügung steht. ■

Umgekehrt kann auch Lock Resolution nicht geordnete Resolution simulieren, da der Index eines Literals unabhängig von seinen Instantiierungen ist.

Für die Grundvollständigkeit wollen wir eine Beweistechnik, vorstellen, die eng verwandt ist mit der im Beweis von Satz 5.9 verwendeten Technik. Der Hauptunterschied ist, daß nicht direkt über die Größe der Signatur induziert wird (was man durchaus tun könnte, aber etwas umständlicher wäre).

Definition 8.24 *Für eine endliche Klauselmenge* S *ist die Zahl der* Überschußliterale *definiert als* $k(S) = (\sum_{C \in S} |C|) - |S|$. ■

Falls $\square \notin S$ gilt $k(S) \geq 0$.

Satz 8.25 *Sei* S *eine endliche Menge indizierter AL-Klauseln. Wenn* S *unerfüllbar ist, dann gilt* $S \vdash_{RL} \square$.

Beweis: Sei $\square \notin S$, sonst ist die Aussage trivial.

Induktion über $k(S)$.

$k(S) = 0$: S muß aus Einerklauseln bestehen. Es muß Klauseln $\{L^i\}$ und $\{\overline{L}^j\}$ in S geben, sonst wäre S erfüllbar (wie ein offener Tableauzweig). $\square$ ist eine Lockresolvente aus $\{L^i\}$ und $\{\overline{L}^j\}$.

$k(S) > 0$: es gibt mindestens eine Klausel $C = \{L^r\} \cup C'$ in S, die keine Einerklausel ist, also $C' \neq \square$.

C ist so gewählt, daß r der höchste Index in den Nicht-Einerklauseln von S ist.

Sei $S_1 = (S - \{C\}) \cup \{C'\}$ und $S_2 = (S - \{C\}) \cup \{\{L^r\}\}$. S_1 und S_2 sind beide unerfüllbar (sonst wäre S erfüllbar). Da die S_i jeweils eine kürzere Klausel als S enthalten, gilt außerdem $k(S_i) < k(S)$. Nach Induktionsvoraussetzung gilt also für $i = 1, 2$ $S_i \vdash_{RL} \square$ mit Resolutionsableitungen Γ_i'.

Γ_1 entstehe aus Γ_1' indem jeweils C an Stelle von C' verwendet wird. Da r der höchste Index in jeder Nicht-Einerklausel von S ist, stellt Γ_1 eine Lockresolutionsableitung dar und endet mit $\{L^r\}$ oder mit $\square$. Falls letzteres der Fall ist, ist der Beweis fertig, bei ersterem hat man immerhin S_2 abgeleitet, und Γ_2' liefert $\square$. ∎

Lemma 8.26 *Für beliebige, indizierte PL-Σ-Klauselmengen S und* $T \subseteq$ $Term^0_\Sigma$ *gilt:*

$$\mathcal{R}_{RL}(r(T(S))) \subseteq T(\mathcal{R}_{RL}(S)) \overset{n.D.}{=} r(T(\mathcal{R}_{RL}(S)))$$

Beweis: Der alte Beweis kann übernommen werden. Es muß lediglich (i) mit reduzierten Grundklauseln gearbeitet werden und (ii) festgehalten werden, daß ein indexminimales Literal in der Grundinstanz $C\rho$ einer indizierten Klausel auch Instanz eines indexminimalen Literals in C ist. Dies garantiert, daß $C\xi_C\nu_C$ und $D\eta_D\mu_D$ Lockfaktoren im Beweis des Lifting-Lemmas sind (Details: Übung). ∎

Beispiel 8.27 Wir wollen die Klauselmenge S aus Beispiel 8.23 zur Illustration des Beweises mit Überschußliteralen verwenden. Eine geeignete Grundinstanz ist

$$S' = \{\{\neg p(\overset{1}{b}, a), p(\overset{2}{a}, b)\}, \{p(\overset{3}{b}, a)\}, \{\neg p(\overset{4}{a}, b)\}\} \ .$$

Das maximale Literal in der einzigen Nichteinerklausel $C = \{\neg p(\overset{1}{b}, a), p(\overset{2}{a}, b)\}$ ist $L^r = p(\overset{2}{a}, b)$. Weiterhin ist $C' = \{\neg p(\overset{1}{b}, a)\}$, und damit

$$S_1 = \{\{\neg p(\overset{1}{b}, a)\}, \{p(\overset{3}{b}, a)\}, \{\neg p(\overset{4}{a}, b)\}\} \ .$$

Aus S_1 kann mit einem Lockresolutionsschritt aus C' und $\{p(\overset{3}{b}, a)\}$ die leere Klausel abgeleitet werden. Ersetzt man C' durch C, erhält man eine Lockresolutionsableitung von $\{L^r\} = \{p(\overset{2}{a}, b)\}$. Aus

$$S_2 = \{\{L^r\}, \{p(\overset{3}{b}, a)\}, \{\neg p(\overset{4}{a}, b)\}\}$$

erhält man nun in einem weiteren Schritt die leere Klausel. ∎

Der größte Nachteil von Lock Resolution ist, daß sie inkompatibel mit vielen anderen Verfeinerungen ist.

Definition 8.28 *Seien* C *und* D *indizierte Klauseln.* C subsumiert D, *falls es eine Substitution* σ *gibt, so daß* $unlock(C)\sigma \subseteq unlock(D)$. C locksubsumiert D, *falls es eine Substitution* σ *gibt, so daß* $C\sigma \subseteq D$. ∎

Proposition 8.29

1. *Lock Resolution mit Tautologielöschung ist unvollständig.*

2. *Lock Resolution mit Vorwärtssubsumtion ist unvollständig.*

3. *Lock Resolution mit Vorwärtslocksubsumtion ist vollständig.*

Beweis: Sei $S = \{\{\overset{1}{p}, \overset{2}{r}\}, \{\overset{3}{\neg r}, \overset{4}{p}\}, \{\overset{5}{r}, \overset{6}{\neg p}\}, \{\overset{7}{\neg p}, \overset{8}{\neg r}\}\}$. S ist unerfüllbar und es gilt

$$\mathcal{R}_{\mathrm{RL}}(S) = \{\{\overset{2}{r}, \overset{8}{\neg r}\}, \{\overset{4}{p}, \overset{6}{\neg p}\}\} \cup S$$

sowie

$$\mathcal{R}^2_{\mathrm{RL}}(S) = \{\{\overset{4}{p}, \overset{8}{\neg r}\}, \{\overset{6}{\neg p}, \overset{8}{\neg r}\}\} \cup \mathcal{R}_{\mathrm{RL}}(S)$$

Aus $\{\overset{4}{p}, \overset{8}{\neg r}\}$ und $\{\overset{7}{\neg p}, \overset{8}{\neg r}\}$ entsteht die Lockresolvente $\{\overset{8}{\neg r}\}$. Daraus erhält man mit $\{\overset{5}{r}, \overset{6}{\neg p}\}$ die Klausel $\{\overset{6}{\neg p}\}$, die mit $\{\overset{1}{p}, \overset{2}{r}\}$ zu $\{\overset{2}{r}\}$ resolviert, woraus man schließlich mit $\{\overset{8}{\neg r}\}$ die leere Klausel gewinnt. Also gilt $\Box \in \mathcal{R}^6_{\mathrm{RL}}(S)$.

1. Mit Tautologielöschung gilt $S = \mathcal{R}_{\mathrm{RL-taut}}(S)$ und also $\Box \notin S = \mathcal{R}^*_{\mathrm{RL-taut}}(S)$.

2. Es wird $\{\overset{4}{p}, \overset{8}{\neg r}\}$ von $\{\overset{3}{\neg r}, \overset{4}{p}\}$ und $\{\overset{6}{\neg p}, \overset{8}{\neg r}\}$ von $\{\overset{7}{\neg p}, \overset{8}{\neg r}\}$ subsumiert. Damit gilt $\Box \notin \mathcal{R}_{\mathrm{RL-fs}}(S) = \mathcal{R}^*_{\mathrm{RL-fs}}(S)$.

3. Übung. Man modifiziere den Beweis von Satz 8.25 in geeigneter Weise.

∎

Historische Bemerkungen: A-Ordnungen wurden unabhängig von Maslov [1964, 1971] (dort etwas allgemeiner und π-Ordnung genannt) und Kowalski and Hayes [1969] eingeführt. Unsere Variante ist die heute üblichste und stammt von Joyner [1976], der als erster geordnete Resolution für Entscheidbarkeitsresultate verwendete. Vorher schon benutzte Maslov einen anderen Kalkül, um ähnliche Resultate zu erhalten.

Fermüller *et al.* [1993]; Leitsch [1997] zeigen Entscheidbarkeitsresultate für weitere Formelklassen.

Lock Resolution wurde von Boyer [1971] eingeführt. Die Idee des Vollständigkeitsbeweises mit Überschußliteralen stammt von Anderson and Bledsoe [1970].

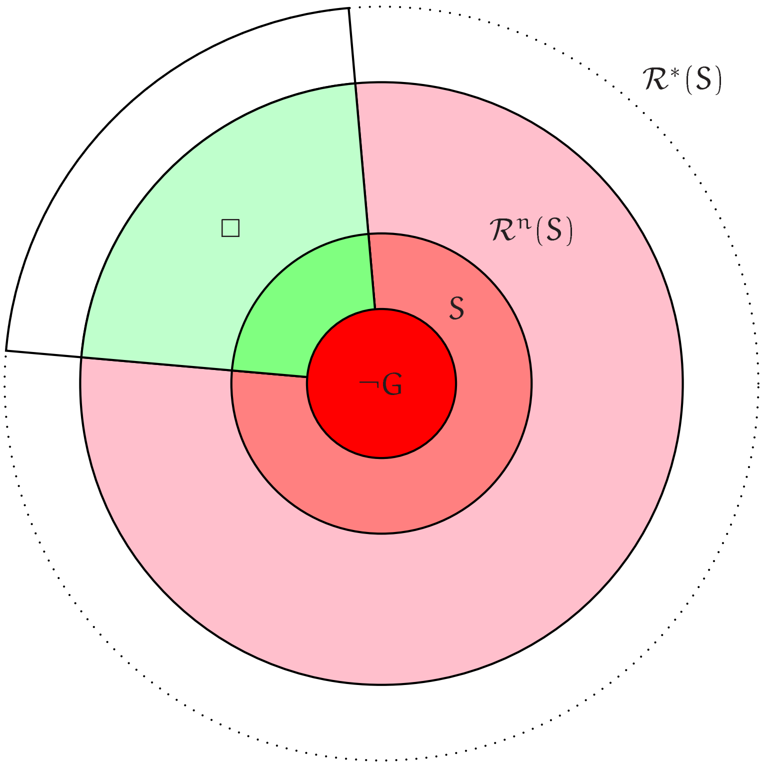


Abb. 8.1: Ein Großteil der ableitbaren Klauseln kann irrelevant sein.

8.3 Einschränkung der Form der Resolutionsableitung

8.3.1 Lineare Resolution

Der größte Nachteil von Resolution ist ihre Ungerichtetheit.

Während bei (schwachen) Konnektionstableaus zuerst die Negation der zu zeigenden Behauptung G zerlegt wird und durch die Konnektionsbedingung dafür gesorgt wird, daß es eine Verbindung zwischen jeder erweiternden Klausel und der ursprünglichen Behauptung gibt, können abgeleitete Resolventen völlig irrelevant für die Erzeugung der leeren Klausel sein.

Wie in Abbildung 8.1 angedeutet, erzeugt die Stufensättigungsstrategie einfach alle möglichen Resolventen aus $S \cup \{\neg G\}$ bis zu einer bestimmten Tiefe. Nur ein kleiner Teil davon muß wirklich relevant für G sein.

Lineare Resolution (LR) ist der Versuch, Resolution ähnlich wie Konnektionstableaus *zielorientierter* zu machen.[3]

Lineare Resolutionsableitungen $\Gamma = \langle D_0, \ldots, D_n \rangle$ müssen eine „direkte Ver-

[3] Historisch gab es lineare Resolution wesentlich früher als Tableaukalküle mit Konnektionen. Das ändert aber nichts an der grundsätzlichen Motivation.

bindung“ von $D_0 \in S$ zu D_n haben.

Dies wird durch die Forderung gewährleistet, daß mindestens eine der Elternklauseln jeder Resolvente entweder die im vorhergehenden Schritt erzeugte Resolvente oder die Startklausel ist.

In der DAG-Darstellung einer Ableitung ergibt sich ein „linearer“ Graph.

Definition 8.30 *Sei S eine Klauselmenge.*

Eine lineare Resolutionsableitung *von D_n aus S ist eine Resolutionsableitung der Form*

$$\Gamma = \langle D_0, D_1, \ldots, D_n \rangle \quad ,$$

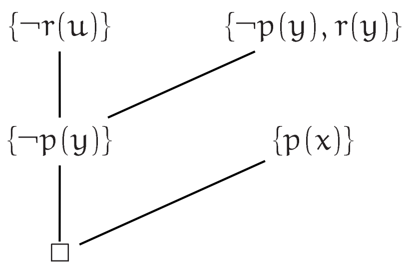
wobei

1. $D_0 \in S$
2. *für alle $i < n$ ist D_{i+1} Resolvente von D_i und $E_{i+1} \in S \cup \{D_0, \ldots, D_i\}$.*

D_0 *heißt* Startklausel *(von Γ), die D_i heißen* Verbindungsklauseln*, die E_i heißen* Seitenklauseln*.* ■

Die Verbindungsklauseln werden manchmal auch *Hauptklauseln* genannt.

Beispiel 8.31 $S = \{\{p(x)\}, \{\neg p(y), r(y)\}, \{\neg r(u)\}, \{q(a)\}\}$. Sei $\{\neg r(u)\}$ die Startklausel. Lineare Resolutionsableitungen werden in DAG-Notation geschrieben, wobei die Vorgängerrelation der Seitenklauseln unterschlagen werden, so daß eine typische lineare Baumform entsteht:



Wie wir schon früher (Beispiel 7.16) gesehen haben, kann man in n Resolutionsschritten aus $\{\neg p(x), p(s(x))\}$ mittels Selbstresolution $\{\neg p(x), p(s^{2^n}(x))\}$ ableiten. Mit linearer Resolution kann man höchstens $\{\neg p(x), p(s^{F(n)}(x))\}$ ableiten, wobei $F(n)$ die n-te Fibonaccizahl ist:

$$\{\neg p(x), p(s(x))\} \qquad \{\neg p(x'), p(s(x'))\}$$

$$\{\neg p(x), p(s^2(x))\}$$

Keine Selbstresolution

$$\{\neg p(x), p(s^3(x))\}$$

$$\{\neg p(x), p(s^5(x))\}$$

$$\vdots$$

$$\{\neg p(x), p(s^{F(n)}(x))\}$$

$F(n)$ n-te Fibonaccizahl

■

Gewisse Ableitungen werden also ausgeschlossen, zum Beispiel ist mit $\{q(a)\}$ als Startklausel in obigem Beispiel kein weiterer linearer Resolutionsschritt möglich. Daran sieht man sofort, daß die Vollständigkeit von linearer Resolution von der Wahl der Startklausel abhängt, in anderen Worten, LR ist nicht beweiskonfluent. Immerhin gilt:

Satz 8.32 *Sei S unerfüllbare AL-Klauselmenge, $C \in S$ aus einer minimal unerfüllbaren (mu) Teilmenge von S. Dann gibt es eine lineare Resolutionsableitung von $\Box$ aus S mit Startklausel C.*

Im folgenden wird ein direkter Beweis angegeben, der auf einer Variation der Überschußliteraltechnik beruht. Später werden wir sehen, daß der Satz auch aus einem allgemeineren Resultat folgt, so daß der Beweis auch übersprungen werden kann.

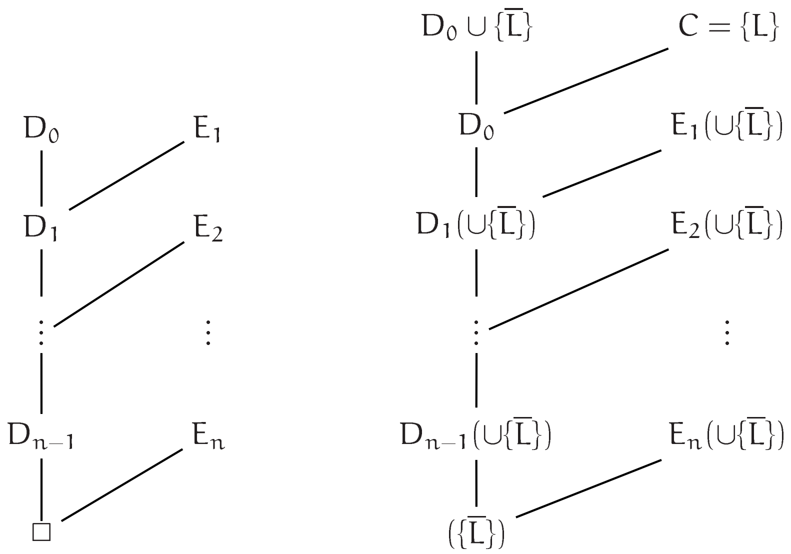
Beweis: Sei S' eine mu von S. Der Satz ist offenbar gezeigt, wenn für alle solchen S' und $C \in S'$ eine lineare Resolutionsableitung von $\Box$ aus S' mit Startklausel C existiert.

Sei O.B.d.A. $\Box$ nicht schon in S'. Wir zeigen per Induktion über $k(S')$, daß für minimal unerfüllbare Klauselmengen S' für alle $C \in S'$ eine lineare Resolutionsableitung von $\Box$ aus S' mit Startklausel C existiert.

$k(S') = 0$: S' muß aus zwei komplementären Einerklauseln bestehen. Beide können als Startklausel für eine lineare Resolutionsableitung von $\Box$ in einem Schritt dienen.

$k(S') > 0$:

Fall 1: $C = \{L\}$ ist eine Einerklausel.



(a) $S^d \vdash \square$ (b) Ableitung von $\{\overline{L}\}$

Abb. 8.2: Illustration zu Fall 1 des Beweises.

$S' - \{C\}$ ist erfüllbar. Für alle Modelle I von $S' - \{C\}$ gilt $val_I(\overline{L}) = W$, sonst hätte man auch ein Modell von S'.

Da S' mu ist, kommt L nur in C vor.

Mit S^d bezeichnen wir die Menge von Klauseln, die entsteht, wenn alle Vorkommen von $\overline{L}$ in $S' - \{C\}$ gelöscht werden. S^d ist unerfüllbar. Da L und $\overline{L}$ in S^d nicht vorkommen, könnte man sonst aus einem Modell für S^d ein Modell für $S' - \{C\}$ mit $val_I(\overline{L}) = F$ machen.

Außerdem ist S^d minimal unerfüllbar. Denn wäre $S^d - \{D\}$ unerfüllbar für ein D, dann auch $S' - \{D\}$ (Übung), aber S' war minimal unerfüllbar.

Wir können annehmen, daß $\overline{L}$ in einer Nicht-Einerklausel in S' auftaucht, sonst wäre $\{L\}$ pur (und S' nicht minimal) oder $\{\overline{L}\} \in S'$ (und S' nicht minimal oder $k(S') = 0$). Also gilt $k(S^d) < k(S')$. Mithin gibt es eine lineare Resolutionsableitung von $\square$ aus S^d (Abb. 8.2(a)) mit einer Startklausel, die in S' noch $\overline{L}$ enthielt.

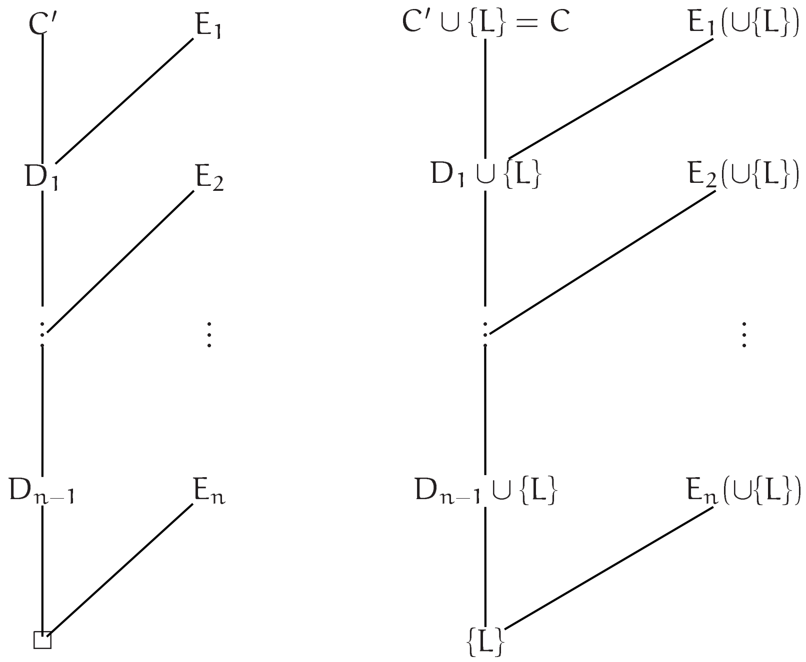
Nun wird zu jeder Klausel $\overline{L}$ hinzugefügt, aus der es entfernt wurde und an den entsprechenden Stellen der Ableitung eingesetzt (Abb. 8.2(b)). D_0 erhält man aus C und $D_0 \cup \{\overline{L}\}$.

Falls die letzte Klausel $\square$ ist, sind wir fertig, ansonsten wird der Beweis mit $\{L\} \in S$ vervollständigt.

Fall 2: $C = \{L\} \cup C'$ ist keine Einerklausel.

$S' - \{C\}$ ist erfüllbar.

Mit S^e bzw. mit $(S' - \{C\})^e$ bezeichnen wir die Mengen von Klauseln, die entstehen, wenn alle Vorkommen von L in S' bzw. in $S' - \{C\}$ gelöscht werden.



(a) $S^{em} \vdash \square$ (b) Ableitung von $\{L\}$

Abb. 8.3: Illustration zu Fall 2 des Beweises.

Es ist offenbar $S^e = (S' - \{C\})^e \cup \{C'\}$. S^e subsumiert S', ist also unerfüllbar.

In jedem Modell I von $S' - \{C\}$ muß $\mathrm{val}_I(L) = F$ gelten, sonst hätte man ein Modell für S'. Da $\mathrm{val}_I(L) = F$, ist I auch Modell von $(S' - \{C\})^e$, welches also erfüllbar ist.

Sei S^{em} eine minimal unerfüllbare Teilmenge von S^e.

$C' \in S^{em}$, sonst ist S^{em} Teilmenge von $(S' - \{C\})^e$ und erfüllbar.

Außerdem ist $k(S^{em}) \leq k(S^e) < k(S')$. Mithin gibt es eine lineare Resolutionsableitung von $\square$ aus S^{em} mit Startklausel C' (Abb. 8.3(a)).

Nun wird zu jeder Klausel L hinzugefügt, aus der es entfernt wurde und eine analoge Ableitung gebildet (Abb. 8.3(b)).

Die Ableitung endet mit $\{L\}$, da über L nicht resolviert wird.

Falls $\{\overline{L}\} \in S'$ sind wir in einem weiteren Schritt fertig.

Ansonsten haben wir eine Ableitung von $\{L\}$ aus S' mit C als Startklausel. Außerdem taucht dann $\overline{L}$ in einer Nicht-Einerklausel $D = \{\overline{L}\} \cup D'$ auf.

Das Argument von eben wird nun für D und $\overline{L}$ wiederholt, wir erhalten also eine lineare Resolutionsableitung von $\square$ aus S^{dm} mit Startklausel D' und daraus eine Ableitung von $\{\overline{L}\}$ oder $\square$ aus S', aber wie im ersten Fall wird $\overline{L}$ nicht zu D' hinzugefügt, so daß D' die Startklausel ist (Abbildung 8.4(a)).

Diese fügen wir an die Ableitung von $\{L\}$ aus S' mit Startklausel C an und erhalten einen linearen Resolutionsbeweis von $\{\overline{L}\}$ oder $\square$ aus S' mit Startklausel

C (Abbildung 8.4(b)).

Falls die letzte Klausel $\square$ ist, sind wir fertig, ansonsten ist noch ein letzter Resolutionsschritt mit $\{L\}$ als Seitenklausel nötig. ■

Lineare Ableitungen für Grundklauseln können ohne weiteres ge„lift"et werden (da dies ein Spezialfall der unbeschränkten Resolution ist), also ist lineare Resolution auch für PL-Klauseln vollständig.

Lineare Resolution ist streng genommen keine Verfeinerung im oben eingeführten Sinn, weil die Möglichkeit eines Resolutionsschrittes von der *Form* der bisherigen Ableitung anstatt von der Menge der Klauseln in der bisherigen Ableitung abhängt. Das Problem ist, daß man eine minimal unerfüllbare Teilmenge, aus der die Startklausel gewählt werden muß, nicht kennt. Eine solche Wahl muß, wie der Beweis zeigt, bei jedem Ableitungsschritt sichergestellt werden.

Das bedeutet, daß LR nicht beweiskonfluent ist und auch nicht über die Stufensättigungsstrategie implementiert werden kann. Vielmehr müssen systematisch alle Ableitungen aufgezählt werden. Es ist genau dieselbe Situation wie bei Konnektionstableaus. Dabei bilden die jeweils als nächstes möglichen Resolutionsschritte die Verzweigungen in einem ODER-Suchbaum.

Für die Implementierung gibt es dementsprechend die schon besprochenen Optionen: Breitensuche, faire Tiefensuche und inkrementelle Tiefensuche. Wie das folgende Beispiel zeigt, ist der Suchraum bei LR enorm.

Beispiel 8.33 Der Suchraum bei LR für die Klauselmenge

$$S = \{\{p, q\}, \{\neg p\}, \{\neg q, r\}, \{\neg r, s\}, \{\neg r, \neg s, t\}, \{p, \neg t\}\}$$

mit Startklausel $\{p, q\}$ ist teilweise in Abbildung 8.5 dargestellt. Die Knoten sind Verbindungsklauseln, an den Kanten sind die verwendeten Seitenklauseln angezeigt. Jeder Zweig entspricht einer linearen Resolutionsableitung. Der vollständige Suchbaum hat über 500 Knoten und mehr als 200 Zweige. Da jeder Zweig, der mit $\square$ endet einer Ableitung $S \vdash \square$ entspricht, ergibt dies für das ziemlich einfache Beispiel schon über 200 verschiedene Beweise! ■

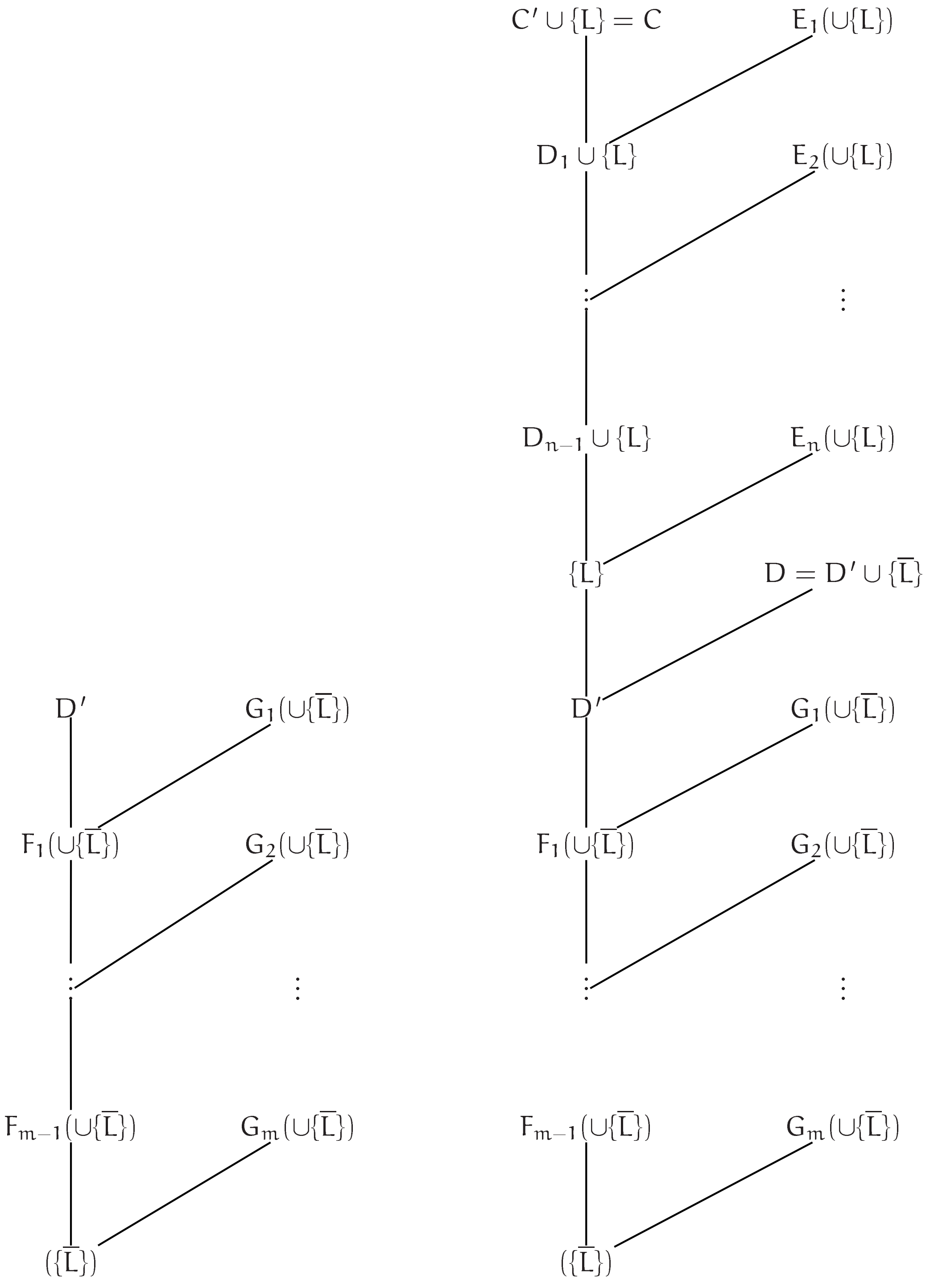
8.3.2 SL-Resolution

Der Suchraum von linearer Resolution ist wie gezeigt wurde sehr groß (viel größer als für unbeschränkte Klauseltableaus); es wurden zahlreiche Versuche unternommen, ihn einzuschränken. Einer der wichtigsten ist die *SL-Resolution* (Linear resolution with Selection function).

In linearen Ableitungen gibt es ja zwei Arten von Resolutionsschritten:

- *Eingabe*[4]*-Resolutionsschritte* im Fall, daß die Seitenklausel aus S ist.

[4] Engl.: input.



(a) Ableitung von $\{\overline{L}\}$.

(b) Ableitung von $\{\overline{L}\}$ und $\{L\}$.

Abb. 8.4: Illustration zu Fall 2 des Beweises.

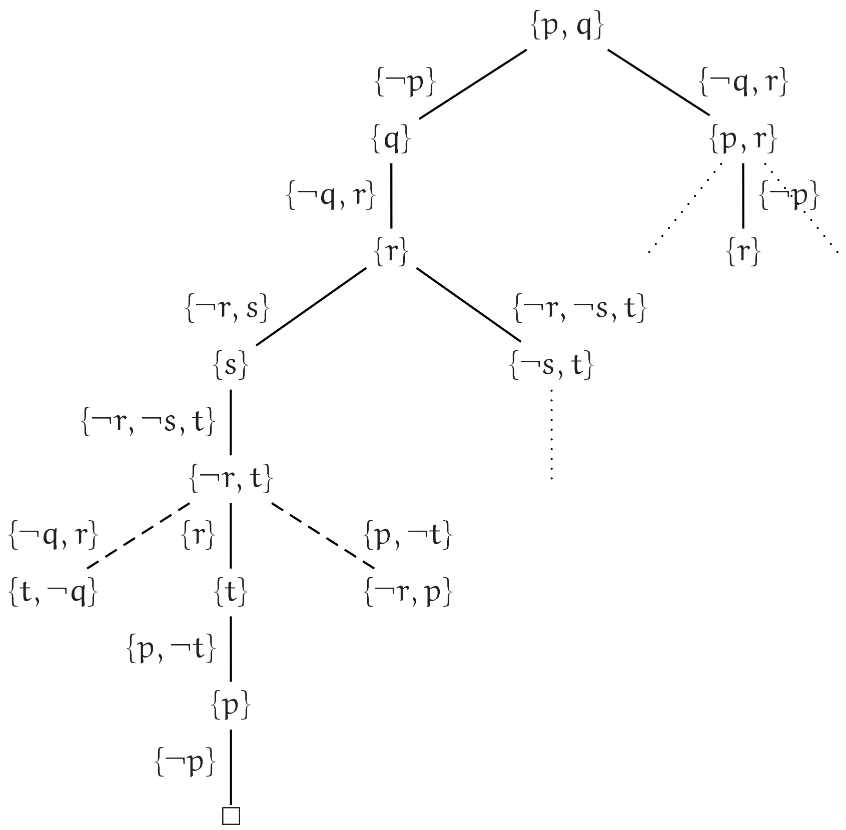


Abb. 8.5: Suchraum bei linearer Resolution.

- *Vorfahr*[5]*-Resolutionsschritte* im Fall, daß die Seitenklausel eine abgeleitete Klausel ist.

Zunächst werden die in Resolutionsschritten verwendbaren Seitenklauseln beschränkt.

Definition 8.34 *Sei C_i eine Verbindungsklausel in einer linearen AL-Resolutionsableitung Γ. Ein Literal $L \in C_i$ ist* Nachfahr *eines Literals L in einer C_i in Γ vorhergehenden Verbindungsklausel C_j gdw L in jedem C_k mit $j \leq k \leq i$ auftaucht. C_j ist ein* A-Vorfahr *von C_i, falls*

1. *die Seitenklausel zu C_j eine Eingabeklausel ist und*
2. *alle Literale in C_j außer dem resolvierten Literal K Nachfahren in C_i haben.*

K heißt dann A-Literal *in C_i vom Vorfahren C_j.* ■

Schematisch sieht das etwa wie folgt aus:

$$\begin{array}{lll}
C_j & \{L_1, \dots, L_m, K\} & \{\overline{K}, \dots\} \\
 & \quad\quad | & \\
C_{j+1} & \{L_1, \dots, L_m, \dots\} & \\
\vdots & \quad\quad \vdots & \\
C_i & \{L_1, \dots, L_m, \dots\} &
\end{array}$$

Hier ist K ein A-Literal in C_i. Die Literale $L_1, \dots, L_m$ sind persistent in $C_j, \dots, C_i$.

Man beachte, daß Einerklauseln mit einer Eingabeklausel als Seitenklausel stets (triviale) A-Vorfahren aller folgenden Verbindungsklauseln sind.

Definition 8.35 *Eine lineare AL-Resolutionsableitung Γ heißt* t-linear *gdw für alle Verbindungsklauseln $C_i \in \Gamma$ folgendes gilt:*

1. *Ancestor-Resolutionsschritte auf C_i sind nur mit A-Vorfahren von C_i möglich.*
2. *Wenn C_i ein zu einem seiner A-Literale komplementäres Literal L enthält, ist C_{i+1} durch Ancestor-Resolution mit L als resolviertem Literal entstanden.*
3. *Die A-Literale von C_i von verschiedenen A-Vorgängern haben verschiedene Atome.*

[5] Engl.: ancestor.

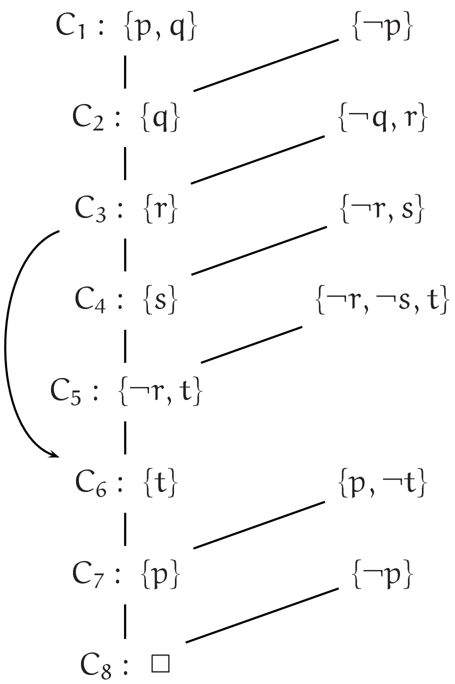


Abb. 8.6: Eine t-lineare Resolutionsableitung.

■

Die zweite Bedingung in obiger Definition bedeutet, daß Ancestor-Resolutionsschritte bevorzugt werden, wann immer sie möglich sind. Die dritte Bedingung entspricht in etwa der Regularität (das wird noch präzisiert werden).

Beispiel 8.36 In Abbildung 8.6 ist eine t-lineare Resolutionswiderlegung von

$$S = \{\{p, q\}, \{\neg p\}, \{\neg q, r\}, \{\neg r, s\}, \{\neg r, \neg s, t\}, \{p, \neg t\}\}$$

dargestellt (vgl. den komplett dargestellten Zweig im Suchbaum aus Abbildung 8.5).

Alle Klauseln außer C_6 sind durch Input-Resolutionsschritte entstanden. Alle Einerklauseln, also C_2, C_3, C_4, sind trivialerweise auch A-Vorfahren von C_5, C_6 usw. Dagegen ist C_1 kein A-Vorfahr von $C_3, \ldots, C_8$, da q keinen Nachfahr in C_3 hat. C_1 ist nur (trivialer) A-Vorfahr von C_2. C_5 ist kein A-Vorfahr von C_6, weil die Seitenklausel zu C_5 keine Eingabeklausel ist. Wegen Bedingung (2) in der Definition von t-Linearität ist der Ancestor-Resolutionsschritt zwingend. Die Knoten unterhalb der gestrichelten Kanten im Suchbaum in Abbildung 8.5 fallen deshalb bei t-linearer Resolution weg.

Die dritte Bedingung der t-Linearität ist ebenfalls erfüllt: der einzige kritische Kandidat ist p in C_1 und C_7, aber C_1 ist kein A-Vorfahr von C_8. ■

Nun werden noch die zu Input-Resolutionsschritten freigegebenen Literale beschränkt.

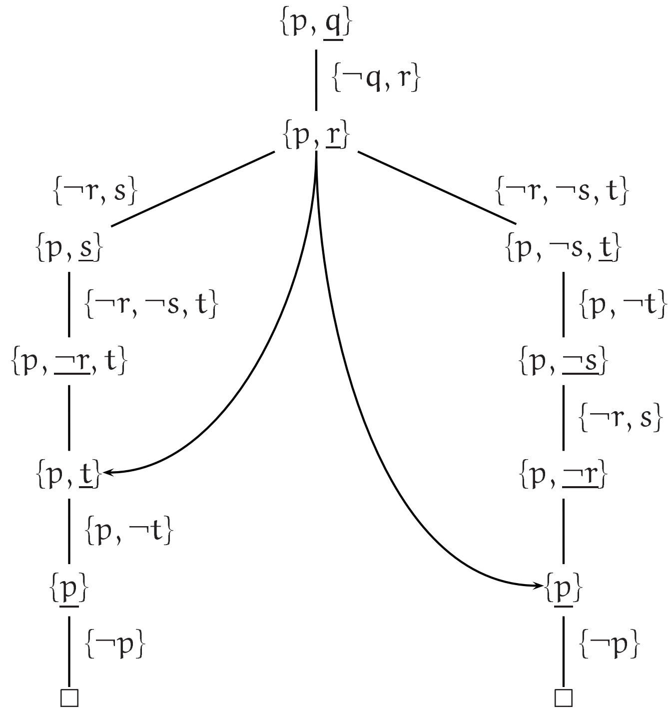


Abb. 8.7: Suchraum von SL-Resolution.

Definition 8.37 *Eine* SL-Auswahlfunktion f *ordnet jeder Verbindungsklausel* C *eines der Literale derjenigen Seitenklausel zu, deren Resolvente* C *ist.*

Eine SL-Resolutionsableitung *ist eine t-lineare Resolutionsableitung, bei der die in Eingaberesolutionsschritten resolvierten Literale durch eine SL-Auswahlfunktion bestimmt sind.* ■

Beispiel 8.38 Bei SL-Resolution mit $f(C) = \mathrm{last}(C)$ hat der gesamte Suchbaum (Abbildung 8.7) für die Klauselmenge S aus Beispiel 8.33 noch zwölf Knoten (bei vorgegebener Startklausel), und es gibt nur zwei unterschiedliche SL-Resolutionswiderlegungen von S. Das dabei resolvierte Literal ist jeweils unterstrichen. Beide Ancestor-Resolutionsschritte sind zwingend. ■

Probleme mit der Definition von SL-Resolution

- Die Definition von SL-Resolution ist kompliziert und schwierig zu verstehen.
- Bei der Erweiterung auf PL hat man verschiedene Optionen bei der Anwendung von Substitutionen. Zunächst ist nicht klar, welche natürlicher ist.

- Eine direkte Implementierung der Definition ist aufwendig.

Alle Probleme lösen sich auf, wenn man *Tableau-* oder *Kettennotation* für SL-Resolutionsbeweise einführt, das heißt es werden nur noch die für SL-Resolution benötigten Informationen gespeichert: die A-Literale der Vorfahren der letzten Klausel und deren unmittelbare Nachfahren.

Voraussetzung für die Kettennotation ist die explizite Trennung von Faktorisierung und Resolution. Bei AL-Resolution ist Faktorisierung durch unsere Mengenschreibweise implizit eingebaut. Bei der an Folgen bzw. Tupel orientierten Kettenschreibweise ist das nicht der Fall.

Definition 8.39 *Sei* C *eine Vielfachmenge und* L *tauche mindestens zweimal in* C *auf. Dann heißt* $C - \{\!\{L\}\!\}$ einfacher Faktor *von* C.

SL_V-Resolution ist wie SL-Resolution definiert, nur daß Klauseln als Vielfachmengen aufgefaßt werden und nach jedem Resolutionsschritt beliebig oft einfach faktorisiert werden darf. Dabei werden die Auftauchen der zuletzt durch einen Input-Resolutionsschritt eingeführten Literale zuerst entfernt. ■

Beispiel 8.40 Sei $C = \{\!\{p, p, \neg q, \neg q, r\}\!\}$. Dann sind $\{\!\{p, \neg q, \neg q, r\}\!\}$, $\{\!\{p, p, \neg q, r\}\!\}$ einfache Faktoren von C, $\{\!\{p, \neg q, r\}\!\}$ dagegen nicht. ■

Beispiel 8.41 Betrachte die Klauselmenge

$$S = \{\{p, q, r\},\ \{\neg p, \neg q\},\ \{\neg p, q, r\},\ \{p, \neg q\},\ \{\neg r\}\}$$

f liefere das am weitesten links stehende Literal in jeder Klausel. Die Startklausel sei $\{p, q, r\}$. Eine SL_V-Resolutionswiderlegung von S ist in Abbildung 8.8 dargestellt. EF steht für „einfache Faktorisierung".

C_1 ist kein A-Vorgänger von C_4, da zwar r, nicht aber q einen Nachkommen in C_4 hat.

Selbst wenn r in C_4 das am weitesten links stehende Literal wäre, könnte es nicht resolviert werden, da es nicht zuletzt in C_4 eingeführt wurde (die SL-Bedingung wäre verletzt).

Die Klausel $\{p, q, r\}$ kann nicht mit C_4 resolviert werden, da sonst die Regularität (die dritte Bedingung der t-Linearität) verletzt würde: die dabei entstehende Klausel ist $C_5' : \{q, r, r\}$; wegen der SL-Bedingung wird unabhängig von der Wahl von f als nächstes das darin vorkommende q resolviert, also hat die entstehende Klausel C_5' als A-Vorgänger mit A-Literal q. Sie hat jedoch auch C_3 mit demselben Literal als A-Vorgänger.

Schließlich ist der Ancestor-Resolutionsschritt, bei dem C_6 entsteht, zwingend, da C_3 ein A-Vorgänger von C_5 ist. ■

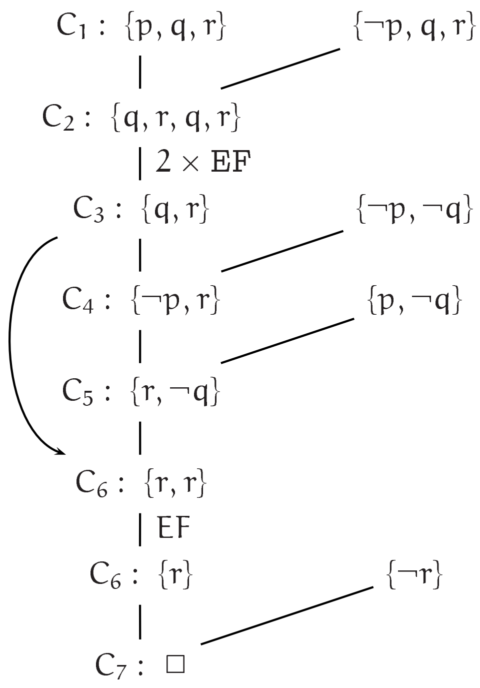


Abb. 8.8: Eine SL_V-Resolutionsableitung.

Offenbar sind SL_V-Resolution und SL-Resolution dahingehend äquivalent, daß jede SL_V-Ableitung leicht in eine SL-Ableitung überführt werden kann und umgekehrt. SL_V-Resolution kann aber nun durch eine Modifikation von schwacher ME beschrieben werden.

Zu jeder endlichen Menge von Literalen C liefert die Auswahlfunktion auf Klauseln f genau ein $f(C) \in C$.

Definition 8.42 zulässige (SL-)Ketten *sind induktiv wie folgt definiert:*

Init: *Jede zu* $C \in S$ *gehörige Kette ist zulässig.*

Extend: *Ist* $\langle L_1, \ldots, L_n \rangle$ *eine zulässige Kette,*

1. $L_r, \ldots, L_n$ *vom Typ* B*,*
2. $\langle M_1, \ldots, M_m \rangle$ *die zu* $C \in S$ *gehörige Kette,*
3. $f(\{L_r, \ldots, L_n\}) = L_k$ *komplementär zu einem* M_j *(*$1 \leq j \leq m$*),*

dann ist auch

$$\langle L_1, \ldots, L_r, \ldots, L_{k-1}, L_{k+1}, \ldots, L_n, \boxed{L_k}, M_1, \ldots, M_{j-1}, M_{j+1}, \ldots, M_m \rangle$$

eine zulässige Kette.

Reduce: *Ist* $\langle L_1, \dots, L_n \rangle$ *eine zulässige Kette,*

1. $L_r, \dots, L_n$ *vom Typ* B,

und ist – ancestor resolution –

2a. $\boxed{L_i}$ *für ein* $i < r$ *vom Typ* A,

3a. $L_i = \overline{L_r}$,

oder – *basic factoring* –

2b. L_i *für ein* $i < r$ *vom Typ* B,

3b. $L_i = L_r$,

dann ist auch

$$\langle L_1, \dots, L_{r-1}, L_{r+1}, \dots, L_n \rangle$$

eine zulässige Kette.

Contract: *Ist* $\langle L_1, \dots, L_n \rangle$ *eine zulässige Kette mit*

$$\boxed{L_j}, \dots, \boxed{L_n} \text{ vom Typ A},$$

dann ist auch

$$\langle L_1, \dots, L_{j-1} \rangle$$

eine zulässige Kette.

Reduce Schritte müssen immer, falls möglich, vor Extend Schritten angewandt werden. Extend Schritte dürfen keine A *Literale produzieren, die schon als* A *Literal vorhanden sind.* ■

Den hieraus resultierenden ME Kalkül nennen wir **Modellelimination mit Selektionsfunktion und Faktorisierung**.

Beispiel 8.43 In Kettennotation sieht die SL_V-Resolutionsableitung aus Abbildung 8.8 wie folgt aus:

p	q	r	Init			
q	r	$\boxed{p}$	q	r	Ext	
q	r	$\boxed{p}$	q	RedF		
q	r	$\boxed{p}$	RedF			
q	r	Contr				
r	$\boxed{q}$	$\neg p$	Ext		(mit $\{p, q, r\}$: r $\boxed{q}$ $\boxed{\neg p}$ q r)[6]	

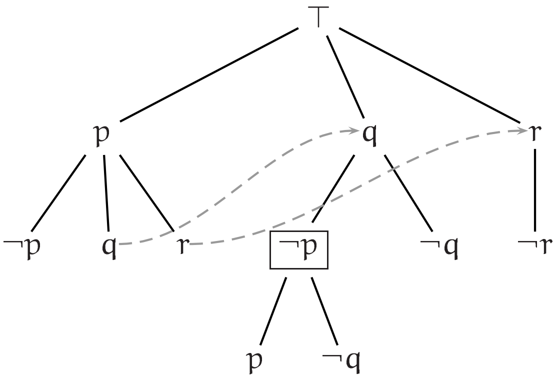


Abb. 8.9: Ein Konnektionstableau mit Faktorisierung, das eine $SL_{\vee}$-Resolutionsableitung modelliert.

$r \quad \boxed{q} \quad \boxed{\neg p}\ \neg q$ Ext
$r \quad \boxed{q} \quad \boxed{\neg p}$ RedA
$r \quad 2\times$ Contr
$\boxed{r}$ Ext
$\langle\rangle$ Contr

Schließlich ist in Abbildung 8.9 noch die reguläre Konnektionstableauvariante (mit Faktorisierng) für dieses Beispiel dargestellt. Die gestrichelten Linien deuten einen Faktorisierungsschritt an. Man beachte, daß das Tableau gemäß einer Left-First Auswahlregel aufgebaut wird, was die spezielle Wahl von f widerspiegelt.

Das umrandete Literal kann natürlich nicht per Abschluß mit dem in einem benachbarten Teilbaum stehenden Komplement kombiniert werden. Bei Resolution muß eine solche Möglichkeit aber explizit ausgeschlossen werden (mit der ersten Bedingung an t-Linearität).

Würde der Zweig, der mit dem umrandeten Literal endet, mit der Klausel $\{p, q, r\}$ erweitert werden, enstünde ein irregulärer Zweig, auf dem q zweimal auftaucht. Im Tableau ist Regularität also sogar etwas stärker als bei t-linearer Resolution, da sie früher erkannt wird. Bei letzterer wäre die dritte Bedingung an t-Linearität erst verletzt, nachdem das zweite q zur Erweiterung (d.h. in einem Input-Resolutionsschritt) verwendet worden wäre. ■

Interpretation von t-Linearität und SL-Bedingung in ME und Tableaus:

t-Linearität 1. Nur A-Literale der letzten Kette bzw. Literale auf dem aktuellen Zweig kommen für die Reduktion in Frage.

[6] Im nächsten Schritt entsteht dann zwangsläufig $r\ \boxed{q}\ \boxed{\neg p}\ \boxed{q}\ r$ und die Kette wird irregulär.

Modellelimination	SL_V-Resolution
Folge von zulässigen Ketten	SL_V-Ableitung
Kette, die mit Typ B Literal endet und Typ A Literale enthält	Verbindungsklausel C
zu $C \in S$ gehörende Kette	Eingabeklausel
ganz rechts stehender Block von Typ B Literalen	Endklausel der Ableitung
Typ B Literal im am weitesten rechts stehenden Block	mögliches Literal für SL-Inputresolution
Typ A Literal in Kette	A-Literal der Endklausel
adjazente Typ B Literale	Literale aus derselben Eingabeklausel
Initiale Kette	Startklausel
Extension	Input-Resolution mit right-first SL-Auswahlfunktion
Reduktion	Ancestor-Resolution
Kontraktion	—
Faktorisierung	einfache Faktorisierung
$\langle\rangle$	$\square$

Tabelle 8.1: Zusammenhang zwischen ME und SL-Resolution.

2. Reduktion und Faktorisierung erfolgen sobald wie möglich.
3. Regularität (von Erweiterungsschritten).

SL-Bedingung Der aktuelle Extensionsschritt muß an einem durch den letzten Extensionsschritt erzeugten Literal ansetzen. Und zwar bei Ketten im am weitesten rechts stehenden Block von B-Literalen, bei Tableau an den Blättern der zuletzt erzeugten offenen Zweige, mit anderen Worten, die Konnektionsbedingung muß erfüllt sein.

SL-Resolution ist damit eine Verstärkung von regulärer Modellelimination (allgemeinere Auswahlfunktion und Faktorisierung).

Nicht t-lineare Schritte in einer Resolutionsableitung entsprechen der Kombination beliebiger Untertableaus.

Der Zusammenhang zwischen ME und SL-Resolution ist noch einmal in Tabelle 8.1 zusammengefaßt.

Für den AL-Fall fügt sich SL-Resolution vollständig in den Rahmen von Konnektionstableaus ein. Insbesondere folgt die Vollständigkeit von SL-Resolution aus der Vollständigkeit von regulären Konnektionstableaus für beliebige Zweigauswahlregeln (und natürlich auch aus der Vollständigkeit von ME).

Daraus folgt die Vollständigkeit der linearen Resolution. Umgekehrt kann der Beweis von Satz 8.32 prinzipiell zu einem Beweis für die Vollständigkeit der

SL-Resolution im AL-Fall und sogar zu einem für ME modifiziert werden. Der Beweis von Satz 5.9 ist aber deutlich einfacher.

Für den prädikatenlogischen Fall hat man verschiedene Optionen bei der Behandlung von Substitutionen:

1. Bei linearer Resolution werden mgus nur auf die Resolvente angewandt also nur auf B-Literale; außerdem werden bei linearer Resolution die Elternklauseln vorher durch Variantenerzeugung variablendisjunkt gemacht.

2. Bei Ancestor-Resolution darf für die Seitenklausel (das eliminierte A-Literal) keine Umbenennung stattfinden, und der mgu bei Reduktion und Extension muß stets auf gesamte Kette/Ableitung/Tableau angewandt werden.

Die zweite Version ist die für ME und Tableau natürliche und (wie wir von dort wissen) für die Vollständigkeit völlig ausreichend, während die erste Version stärker als notwendig ist und den Suchraum unnötig vergrößert.

Zwar wurde sie für SL-Resolution schon bei ihrer Einführung so gewählt, allerdings ohne den Tableaubezug sehr schwer zu motivieren.

Die Implementierung erfolgt in jedem Fall mit denselben Techniken wie die von Konnektionstableaus.

8.3.3 Input-Resolution

Definition 8.44 *Sei* S *eine Klauselmenge. Eine* Input-Resolutionsableitung *aus* S *ist eine Resolutionsableitung aus* S*, in der jede Resolvente mindestens eine Eingabeklausel als Elternklausel hat.* ■

Input-Resolution ist lineare Resolution ohne Ancestor-Resolution. Input-Resolution ist für allgemeine Klauselmengen unvollständig.

Beispiel 8.45 $C_2 = \{\{p, q\}, \{\neg p, q\}, \{p, \neg q\}, \{\neg p, \neg q\}\}$. In C_2 haben alle Eingabeklauseln die Länge 2. Jede Verbindungsklausel hat mindestens die Länge 1. Also hat auch jede Resolvente mindestens die Länge 1, woraus folgt, daß man die leere Klausel nicht ableiten kann. ■

Dagegen gilt:

Satz 8.46 *Sei* S *eine unerfüllbare Menge von Hornklauseln. Dann gibt es eine Input-Resolutionsableitung von* $\Box$ *aus* S*.*

Der Beweis erfolgt später in Satz 8.55.

8.3.4 Unit-Resolution

Definition 8.47 *Ein* Unit-Resolutionsschritt *ist ein Resolutionsschritt, bei dem mindestens eine(r) der (Faktoren der) Elternklauseln eine Einerklausel ist. Eine* Unit-Resolutionsableitung *ist eine Resolutionsableitung, in der jeder Resolutionsschritt ein Unit-Resolutionsschritt ist.* ■

Im Gegensatz zu Input-Resolution ist Unit-Resolution eine Resolutionsverfeinerung im engeren Sinn, das heißt, man kann sie durch einen Verfeinerungsoperator, wie am Anfang dieses Kapitels eingeführt, charakterisieren.

Satz 8.48 *Für PL-Klauselmengen S gilt: $\Box$ ist aus S mit Unit-Resolution ableitbar gdw $\Box$ aus S mit Input-Resolution ableitbar ist.*

Der Beweis (siehe etwa [Chang and Lee, 1973, S. 134f]) verwendet ähnliche Argumente wie der Vollständigkeitsbeweis der linearen Resolution bzw. der Lockresolution.

Exkurs: Komplexität von Hornformeln

Satz 8.49 *Die Erfüllbarkeit einer Menge AL-Hornformeln kann in linearer Zeit entschieden werden.*

Wie schon früher führen wir eine besondere Terminologie für Hornklauseln ein.

Negative Hornklauseln $\{\neg p_1, \ldots, \neg p_n\}$ werden $\Box \leftarrow p_1, \ldots, p_n$ geschrieben; positive Hornklauseln (Fakten) $\{p\}$ werden $p \leftarrow \top$ geschrieben; gemischte Hornklauseln (Regeln) $\{p, \neg p_1, \ldots, \neg p_n\}$ werden $p \leftarrow p_1, \ldots, p_n$ geschrieben.

Weiter seien für alle Hornklauseln $C = p \leftarrow p_1, \ldots, p_n$ Funktionen

$$\mathrm{Head}(C) = p \text{ und } \mathrm{Body}(C) = \{p_1, \ldots, p_n\}$$

erklärt.

Beweis: Zu einer AL-Hornformel $S = \{C_1, \ldots, C_m\}$ wird ein gerichteter, markierter Graph $G_S = \langle N, E \rangle$ definiert mittels

$$N = \Sigma_S \cup \{\top, \Box\}$$

$$E = \{\langle p, q, C \rangle \mid p, q \in N, \text{ es gibt } C \in S, \text{ so daß } q = \mathrm{Head}(C), p \in \mathrm{Body}(C)\}$$

$\langle p, q, C \rangle$ heißt, daß es in G_S eine Kante von p nach q mit Markierung C gibt.

Die Tiefe d eines Knotens in G_S ist $d(\top) = 0$ und für $p \neq \top$:

$$d(p) = 1 + \min\{d_0(p, C) \mid \text{ es gibt } \langle p', p, C \rangle \in E\}$$

Hierbei wird festgelegt: $\min\{\} = \omega$ und

$$d_0(p, C) = \max\{d(p') \mid \langle p', p, C\rangle \in E\} \ .$$

p heißt *erreichbar*, falls $d(p) < \omega$.

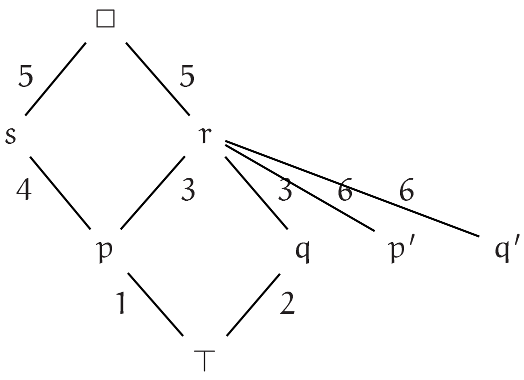
Es gilt: wenn p erreichbar ist, dann $S \models p$. Beweis: Übung (Hinweis: Induktion über $d(p)$). Damit folgt, daß S unerfüllbar ist gdw $\Box$ erreichbar ist.

G_S kann in $\mathcal{O}(|S|)$ Schritten aufgebaut werden. Die Erreichbarkeit eines Knotens in G_S kann mit einem *Bottom-up*-Markierungsalgorithmus in linearer Zeit getestet werden. ■

Beispiel 8.50 Gegeben sei die Hornklauselmenge

$$S = \{p \stackrel{1}{\leftarrow} \top \mid q \stackrel{2}{\leftarrow} \top \mid r \stackrel{3}{\leftarrow} p, q \mid s \stackrel{4}{\leftarrow} p \mid \Box \stackrel{5}{\leftarrow} s, r \mid r \stackrel{6}{\leftarrow} p', q'\}$$

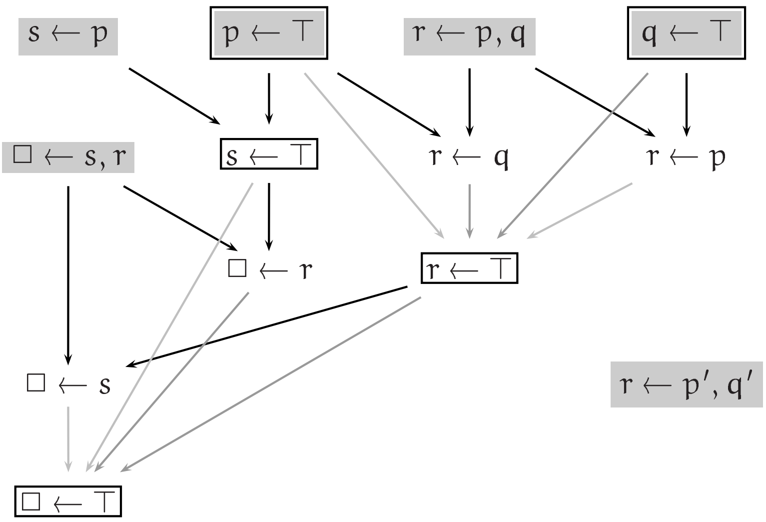
$N = \{p, q, r, s, p', q', \Box, \top\}$. Der Erfüllbarkeitsgraph G_S sieht aus wie folgt:



Die Tiefe beispielsweise des Knotens r berechnet man wie folgt:

$d(r) = 1 + \min\{d_0(r, 3),\ d_0(r, 6)\}$
$d_0(r, 3) = \max\{d(p),\ d(q)\}$
$d_0(r, 6) = \max\{d(p'),\ d(q')\}$
$d(p) = 1 + d_0(p, 1) = 1 + d(\top) = 1$
$d(q) = 1 + d_0(q, 2) = 1 + d(\top) = 1$
$d_0(r, 3) = 1$
$d(p') = 1 + \omega = \omega$
$d_0(r, 6) = \omega$
$d(r) = 1 + \min\{1, \omega\} = 2$

Eine Unit-Resolutionsableitung, die sich an der Definition von d im obigen Beweis orientiert und sukzessiv $D_i(S) = \{p \mid S \models p,\ d(p) = i\}$ für $i = 1, 2, \ldots$ berechnet, sieht so aus:



Eingabeklauseln sind unterlegt; Klauseln, die Knoten im Erfüllbarkeitsgraphen entsprechen, sind gerahmt. ■

Unit-Resolution entspricht mithin einem Markierungsalgorithmus für den oben eingeführten Erfüllbarkeitsgraphen G_S einer Hornformel S. Dies bedeutet, daß Unit-Resolution mit entsprechenden Datenstrukturen einen linearen Erfüllbarkeitstest für Hornformeln darstellt, siehe Kleine Büning and Lettmann [1994].

Der Exkurs endet hier.

Unit-Resolution kann zur heuristischen Verbesserung einer vollständigen Resolutionsverfeinerung verwendet werden.

Zum Beispiel können vor jedem unbeschränkten Resolutionsschritt i-ter Tiefe alle möglichen Unit-Resolventen erzeugt werden, die in höchstens i Schritten abgeleitet werden können. Mit

$$\begin{aligned}
\mathcal{R}^0_{RU}(S) &= S \\
\mathcal{R}^{i+1}_{RU}(S) &= \mathcal{R}_{RF}(\underbrace{\mathcal{R}_U \cdots \mathcal{R}_U}_{i \text{ mal}}(\mathcal{R}^i_{RU}(S))) \cup \mathcal{R}^i_{RU}(S) \\
\mathcal{R}^*_{RU}(S) &= \bigcup \mathcal{R}^i_{RU}(S)
\end{aligned}$$

ist $\mathcal{R}^*_{RU}$ vollständig (Übung) und liefert oftmals bessere Ergebnisse als $\mathcal{R}^*_{RF}$.

Der Effekt ist, daß Unit-Resolutionsschritte früher stattfinden als in $\mathcal{R}^*_{RF}$, was kürzere Klauseln erzeugt und Subsumtion wahrscheinlicher macht. Dies ist besonders wirkungsvoll, wenn der Hornklauselanteil groß ist. $\mathcal{R}_{RU}$ heißt *Unit-Preference Strategie*.

8.3.5 SLD-Resolution

Definition 8.51 *Ein* N-(P-)Resolutionsschritt *ist ein Resolutionsschritt, bei dem mindestens eine der Elternklauseln eine negative (positive) Klausel ist. Eine* N-(P-)Resolutionsableitung *ist eine Resolutionsableitung, in der jeder Resolutionsschritt ein N-(P-)Resolutionsschritt ist.* ∎

Diese Verfeinerung wird sich im nächsten Abschnitt als ein Spezialfall der sogenannten Hyperresolution (semantischen Resolution) erweisen.

Für Hornformeln gilt: *jede* Resolutionsableitung mit einer negativen Startklausel ist sowohl N- als auch Input-Resolutionsableitung, in der nur definite (nichtnegative) Klauseln als Eingabeklauseln vorkommen.

Für Input-Resolution ist t-Linearität nicht definiert (Definition 8.35). Man kann aber die SL-Bedingung (Definition 8.37) mit der Input-Bedingung kombinieren:

Definition 8.52 *Eine* SLD-Resolutionsableitung *ist eine Ableitung, die zugleich eine N- und eine Input-Resolutionsableitung ist und die SL-Bedingung erfüllt.* ∎

Eine SLD-Resolutionsableitung aus Hornklauseln ist also einfach eine Ableitung mit negativer Startklausel und SL-Auswahlfunktion. SLD-Resolution ist natürlich i.a. unvollständig. SLD-Resolution steht für: SL-Resolution für definite Klauseln.

Im Logischen Programmieren (Apt and van Emden [1982]; Lloyd [1987]) wird die operationale Semantik von Prolog Programmen oft ebenfalls durch SLD-Resolution charakterisiert. Dort meint man aber meist etwas anderes, das wir Prolog-SLD-Resolution nennen wollen (in der Literatur auch als LUSH-Resolution bezeichnet):

Definition 8.53 *Eine* Prolog-SLD-Resolutionsableitung *ist eine Ableitung, die gleichzeitig N- und Input-Resolutionsableitung ist, bei der die resolvierten Literale durch eine Auswahlfunktion auf Klauseln bestimmt sind und worin Klauseln als Vielfachmengen repräsentiert sind.* ∎

Die Standard-Prolog-Auswahlfunktion wählt das am weitesten links stehende Literal aus, womit für Prolog Programme auch die SL-Bedingung erfüllt ist, falls Klauseln als Tupel behandelt werden und neue Literale links angehängt werden. Die laxe Bezeichnung der SLD-Resolution ist deshalb vetretbar. Man beachte, daß es in Prolog-SLD-Resolution keine Faktorisierung gibt, was für allgemeine PL-Klauselmengen ja unvollständig ist.

Beispiel 8.54 Im Prolog Beweissuchbaum für die Hornklauselmenge aus Beispiel 3.20 (Abbildung 3.15) repräsentiert der mittlere Zweig sowohl eine SLD-Widerlegung als auch eine Prolog-SLD-Widerlegung mit left-first Auswahlfunktion (falls das Cut Prädikat ignoriert wird). Man beachte, daß in Prolog-SLD-Resolution auch eine Auswahlfunktion zulässig wäre, die zum Beispiel in der Klausel $\leftarrow s(x), t(x), r$ das Literal r auswählt, die SL-Bedingung aber dadurch verletzt würde und dies somit keine SL-Resolutionsableitung darstellt.

Während mit SLD-Resolution aus der Klauselmenge $\{\leftarrow p, p \leftarrow q, q\}$ die Klausel $\{\leftarrow q\}$ abgeleitet wird, erhält man bei Prolog-SLD-Resolution $\{\leftarrow q, q\}$; die Widerlegung von q muß auch zweimal erfolgen.

Natürlich kann man Prolog-Beweissuchbäume auch in Kettennotation repräsentieren. Für unser obiges Beispiel ergibt sich:

$\neg p(x)$ Init

$\boxed{\neg p(x)}$ $\neg q(x)$ $\neg r$ Ext

$\boxed{\neg p(x)}$ $\neg r$ $\boxed{\neg q(x)}$ $\neg s(x)$ $\neg t(x)$ Ext

$\boxed{\neg p(c)}$ $\neg r$ $\boxed{\neg q(c)}$ $\boxed{\neg s(c)}$ $\neg t(c)$ Ext

$\boxed{\neg p(c)}$ $\neg r$ $\boxed{\neg q(c)}$ $\boxed{\neg s(c)}$ $\boxed{\neg t(c)}$ Ext

$\boxed{\neg p(c)}$ $\neg r$ Contr

$\boxed{\neg p(c)}$ $\boxed{\neg r}$ Ext

$\langle\rangle$ Contr

Man beachte, daß die Substitutionen auf die gesamte Kette angewandt werden. Die Reduktionsregeln (entsprechend Faktorisierung und Ancestor-Resolution) dürfen nicht verwendet werden. Da im Prolog Beweissuchbaum i.a. left-most erweitert wird, wäre es natürlicher die Ketten von rechts nach links wachsen zu lassen. Dann wird einfach das am weitesten links stehende Literal selektiert und, da Reduktion nicht möglich ist, kann auf die Repräsentation der B-Literale verzichtet werden. Mit diesen Modifikationen entspricht eine Kettenableitung gerade einem Pfad in einem Beweissuchbaum. ∎

Satz 8.55 *Sei S eine AL-Hornformel. Ist S unerfüllbar, dann gibt es eine Prolog-SLD-Resolutionsableitung von $\square$ aus S.*

Beweisskizze: Induktion über die Anzahl n der Literale in S.

$n = 2$: $S = \{\{\!\{p\}\!\}, \{\!\{\neg p\}\!\}\}$. Klar.

$n > 2$: S sei o.B.d.A. minimal unerfüllbar. Jede unerfüllbare AL-Hornformel enthält eine positive Einerklausel $\{p\}$ (alle positiven Hornklauseln sind Einerklauseln, falls nicht enthält also jede Klausel ein negatives Literal und S ist erfüllbar).

Falls $\{\neg p\} \in S$, ist man fertig. Sonst entstehe S^d aus S, indem jedes Auftauchen von $\neg p$ in S gelöscht wird. Da $\{\neg p\} \notin S$ ist $\Box \notin S^d$; außerdem ist S^d unerfüllbar.

$\neg p$ taucht wenigstens einmal in S auf, sonst ist $S - \{\{p\}\}$ unerfüllbar und S nicht mU. Nach Induktionsvoraussetzung gibt es daher eine Prolog-SLD-Resolutionsableitung von $\Box$ aus S^d. Wir setzen in die Klauseln dieser Ableitung $\neg p$ wieder ein. Dadurch endet die Ableitung nicht mehr mit $\Box$, sondern mit $\{\neg p\} = \{\!\{\neg p, \ldots, \neg p\}\!\}$. Eine entsprechende Anzahl Resolutionsschritte mit $\{p\} \in S$ ergibt eine Prolog-SLD-Resolutionsableitung von $\Box$ aus S. ■

Korollar 8.56 *SLD-, Input-, Unit-Resolution sind widerlegungsvollständig für AL-Hornformeln.*

Das Lifting von SL- und SLD-Ableitungen ist problemlos und sogar etwas einfacher als für unbeschränkte Resolution, da eingeschränkte bzw. keine Faktorisierung vorkommt und die Position des resolvierten Literals durch die SL-Bedingung genauer bestimmt ist.

Historische Bemerkungen: Lineare Resolution wurde – unabhängig – von Loveland [1968] und Luckham [1968] vorgeschlagen.

SL-Resolution stammt von Kowalski and Kuehner [1971], die auch die Ähnlichkeit zur schwachen ME, nicht jedoch zu Konnektionstableau, darlegten. SL-Resolution und der Zusammenhang zu ME wurde auch, unabhängig davon, von Loveland [1972] gefunden, SL-Resolution alleine auch noch von Reiter [1971].

Kowalski and Kuehner [1971] diskutierten Suchstrategien und schlugen eine Interpretation von SL-Resolution als Problemzerlegung vor (die Auswahlfunktion „richtet" die Klauseln), welche die Entwicklung von Prolog entscheidend beeinflußt hat.

Unit-Resolution ist im wesentlichen von Davis and Putnam [1960]; unsere Unit-Preference Strategie ist eine vereinfachte Version von Wos *et al.* [1964]. Der Zusammenhang von Unit- und Input-Resolution wurde von Chang [1970] entdeckt.

Der erste Beweis, daß die Erfüllbarkeit von Hornformeln in linearer Zeit entschieden werden kann, steht bei Dowling and Gallier [1984]. Eine verbesserte Version, in der auch die Erfüllbarkeitsgraphen eingeführt werden, ist bei Gallo and Urbani [1989] zu finden.

SLD-Resolution wurde von Kowalski [1974] definiert und von Apt and van Emden [1982] so bezeichnet. Dort wurde tatsächlich aber Prolog-SLD-Resolution verwendet. Letztere heißt LUSH-Resolution bei Hill [1974], der auch ihre Vollständigkeit bewies.

8.4 Makro-Resolutionsschritte

8.4.1 Semantische Resolution

Die AL-Resolutionsregel hat eine interessante Eigenschaft:

Sei S eine unerfüllbare Klauselmenge und I eine *beliebige* AL-Interpretation, zum Beispiel $S = \{\{\neg p, q\}, \{p, q, \neg r\}, \{\neg s, t\}, \{\neg s, \neg t\}, \{r, p\}, \{\neg q, s\}\}$ und $I = \{q\}$.[7]

Es ist stets möglich, Klauseln C, D aus S zu finden, so daß gilt:

- $I \not\models C$;
- $I \models D$;
- C und D sind AL-resolvierbar.

Denn sei L ein Literal, das in einer minimal unerfüllbaren Teilmenge S' von S vorkommt und zwar in $C \in S'$ mit $I \not\models C$. So ein C existiert, weil S' unerfüllbar ist. Wegen der Minimalität ist auch $D \in S'$ mit $\overline{L} \in D$.

Zum Beispiel resolvieren $C = \{\neg q, s\}$, $D = \{\neg s, t\}$ zu $\{\neg q, t\}$. Hier ist $L = s$.

Man beobachtet, daß sogar $I \not\models \{\neg q, t\}$. Im allgemeinen braucht das für einen einzelnen Resolutionsschritt zwar nicht zu gelten, aber durch mehrere geeignete Schritte läßt es sich immer erreichen.

Definition 8.57 *Ein* Clash *ist eine Menge von Klauseln* $\{E_1, \ldots, E_q, C\}$ *mit* $|C| \geq q > 0$*, so daß*

1. $L_1, \ldots, L_q \in C$;
2. *Für alle* $1 \leq i \leq q$ *gilt* $\overline{L_i} \in E_i$
3. *Für alle* $1 \leq i, j \leq q$ *gilt:* $(C - \{L_i\}) \cup E_i$ *und* $E_i \cup E_j$ *ist nicht tautologisch.*

C *heißt der* Kern[8] *des Clashs, die* E_i *heißen* Elektronen. ■

Definition 8.58 *Sei* $T = \{E_1, \ldots, E_q, C\}$ *eine AL-(PL-)Klauselmenge. Falls eine lineare AL-(PL-)Resolutionsableitung* $\langle C, E_1, D_1, \ldots, E_q, D_q \rangle$ *existiert – mit* q *Schritten (und mgus* $\sigma_1, \ldots, \sigma_q$*) –, so daß* T *(bzw.* $T\sigma_1 \cdots \sigma_q$*) ein Clash ist, dann heißt* D_q Clash-Resolvente *von* T. ■

[7] Wie schon früher werden Interpretationen mit der Menge der Atome, die sie wahr machen, identifiziert.

[8] Engl.: nucleus.

Beispiel 8.59 In den folgenden Mengen seien jeweils die ersten beiden Klauseln Elektronen, die dritte Klausel der Kern. $T_1 = \{\{r, p\}, \{\neg q, s\}, \{\neg p, q\}\}$ ist ein Clash mit der Resolvente $\{r, s\}$; $T_2 = \{\{r, p\}, \{\neg q, s\}, \{p, q, \neg r\}\}$ ist ein Clash mit der Resolvente $\{p, s\}$; $T_3 = \{\{\neg p, q\}, \{\neg q, s\}, \{p, q, \neg r\}\}$ ist kein Clash, da q und sein Komplement in den Elektronen auftaucht. Mindestens eines der resolvierten Literale würde dadurch wieder in die Resolvente eingeführt. ∎

Im folgenden fassen wir Herbrand-Interpretationen I als maximal konsistente Mengen von Grundliteralen über der Signatur einer PL-Klauselmenge auf. Jede solche Interpretation entspricht gerade einem Zweig in einem semantischen Baum über Σ.

Definition 8.60 *Das* Fehlschlagen *einer Klausel* C *in einer Interpretation* I *ist genau wie für die Widerlegungsmengen von semantischen Bäumen definiert: es gibt eine Grundsubstitution* σ*, so daß für jedes Literal* $L \in C\sigma$ *das Komplement* $\overline{L}$ *von* I *wahr gemacht wird.* ∎

Definition 8.61 *Es seien* $T = \{E_1, \ldots, E_q, C\}$ *AL-(PL-)Klauseln und* D *eine Clash-Resolvente von* T*.* I *ist eine AL-(Herbrand-)Interpretation für* T*, so daß* $\{E_1, \ldots, E_q\}$ *in* I *fehlschlagen. Dann ist* T *ein* I-Clash *und* D *eine* I-Clash-Resolvente*. Falls* D *in* I *fehlschlägt, ist* T *ein* maximaler I-Clash*.* ∎

Aus der Bedingung $I \not\models E_1, \ldots, I \not\models E_q$ folgt die erste Clash-Bedingung, nämlich daß $E_i \cup E_j$ für kein i, j tautologisch ist.

In jeder I-Clash-Resolvente liefert I ein Modell des Kerns C des Clashs. Jede maximale Resolvente kann als Elektron in folgenden I-Clash-Schritten verwendet werden.

Definition 8.62 *Eine* I-Clash-Resolutionsableitung *ist eine Ableitung, deren jeder Schritt ein* I*-Clash-Resolutionsschritt ist.* ∎

Beispiel 8.63 Seien S und I wie im vorhergehenden Beispiel. Die Clashs T_1 und T_2 im vorhergehenden Beispiel sind maximale I-Clashs.

Die maximale I-Clash-Resolvente $\{p, s\}$ von T_2 liefert mit $\{\neg p, q\}$ als Kern und $\{\neg q, s\}$ die maximale I-Clash-Resolvente $\{s\}$. Daraus erhält man mit $\{\neg s, t\}$ als Kern $\{t\}$. Aus $\{t\}$ und $\{s\}$ mit $\{\neg s, \neg t\}$ als Kern $\square$. Insgesamt hat man eine maximale I-Clash-Widerlegung von S. ∎

Satz 8.64 *Ist* S *eine unerfüllbare Menge von PL-Klauseln, so ist* $\square$ *aus* S *mit* I*-Clash-Resolution für beliebige Herbrand Interpretationen* I *ableitbar.*

Wie üblich werden AL-Interpretationen der Grundinstanzen von S und Herbrand-Interpretationen auf S miteinander identifiziert.

Lifting geht wie üblich mit der Beobachtung, daß jeder AL-I-Clash auf einer Grundmenge T auch ein PL-I-Clash auf der ge„lift“eten Menge T' ist. Diesen kann man entlang der Ableitung in Definition 8.58 zusammensetzen. Hierfür braucht man die früher erwähnte allgemeinere Version des Lifting-Lemmas für beliebige Instanzen (nicht notwendigerweise Grundinstanzen), weil die „Zwischenergebnisse“ in der linearen Ableitung nicht grund sein müssen.

Die Grundvollständigkeit wird aus einem später gezeigten allgemeineren Resultat folgen.

Wählt man jeweils minimale Clashs (für ein beliebiges I), also $q = 1$, dann hat man die Vollständigkeit der normalen Resolution.

8.4.2 UR-Resolution

Definition 8.65 *Ein* UR-Resolutionsschritt *ist ein maximaler Clash-Resolutionsschritt, bei dem die Resolvente entweder eine Einerklausel oder die leere Klausel ist.*

Eine UR-Resolutionsableitung *ist eine Resolutionsableitung, in der jeder Resolutionsschritt ein UR-Resolutionsschritt ist.* ■

UR-Resolution ist nicht vollständig für allgemeine Klauseln, ist es aber für Hornformeln. Der Vorteil gegenüber der Unit-Resolution ist:

- Die Reihenfolge der Unit-Resolutionsschritte, aus denen sich der Clash-Resolutionsschritt zusammensetzt ist irrelevant. Dies verhindert Resolutionsableitungen, die sich nur hinsichtlich der Reihenfolge ihrer Schritte unterscheiden.
- Die Zwischenergebnisse können keine zusätzlichen Resolventen erzeugen.

Beispiel 8.66 Betrachte

$$S = \{\overset{C_1}{\{\neg p(x,y), \neg p(y,z), p(x,z)\}}, \overset{C_2}{\{p(a,b)\}}, \overset{C_3}{\{p(b,c)\}}\}$$

worin C_1 die Transitivität des Prädikats p repräsentiert; ähnliche Axiome sind häufig Teil einer Axiomatisierung. Mit unbeschränkter Resolution können sowohl $\{\neg p(b,z), p(a,z)\}$ (aus C_1 und C_2) als auch $\{\neg p(x,b), p(x,c)\}$ (aus C_1 und C_3) als Resolventen auftreten. Beide resolvieren mit C_2 beziehungsweise mit C_1 zu $\{p(a,c)\}$. Bei UR-Resolution mit $I \equiv F$ ist letzteres die einzig mögliche Resolvente. Die irrelevanten Zwischenergebnisse werden gar nicht erst erzeugt.
■

8.4.3 Hyper-Resolution

Die Interpretation $I_F \equiv F$ (und ihr Dual) sind wichtige Spezialfälle der semantischen Resolution falls die Unit-Einschränkung weggelassen wird.

Sei also I_W diejenige Interpretation für S, die alle Grundinstanzen von Atomen aus S wahr macht, $I_W = At^0_\Sigma$; $I_F = \{\}$ dagegen macht alle Grundinstanzen von Atomen aus S falsch.

In einem maximalen I_F-Clash $\{E_1, \ldots, E_q, C\}$ sind also alle E_i positive Klauseln, und die negativen Literale von C sind gerade die resolvierten Literale, nämlich $\{L_1, \ldots, L_q\}$. Die Resolvente ist wieder positiv.

Dieser Spezialfall von semantischer Resolution ist als Hyper-Resolution bekannt:

Definition 8.67 *Ein* positiver (negativer) Hyper-Resolutionsschritt *ist ein maximaler* I_F*-(*I_W*-)Clash-Resolutionsschritt.*

Eine positive (negative) Hyper-Resolutionsableitung *ist eine Resolutionsableitung, in der jeder Schritt ein positiver (negativer) Hyper-Resolutionsschritt ist.* ■

Die Vollständigkeit folgt aus derjenigen der semantischen Resolution.

Hyper-Resolution läßt sich auch als Einschränkung von P- bzw. von N-Resolution deuten, da jede positive (negative) Hyper-Resolutionssableitung auch P-(N-)Ableitung ist.

Beispiel 8.68 Hyper-Resolution ist inkompatibel mit linearer Resolution, wie man aus folgendem Beispiel ersieht:

Sei $S = \{\{p\}, \{\neg p, q\}, \{\neg q, \neg r\}, \{r, \neg p\}\}$. Da $\{p\}$ die einzige positive Klausel ist, muß jede positive Hyper-Resolutionsableitung sie als einziges Elektron im ersten Schritt benutzen. Damit sind nur $\Gamma_1 = \langle\{p\}, \{\neg p, q\}, \{q\}\rangle$ und $\Gamma_2 = \langle\{p\}, \{r, \neg p\}, \{r\}\rangle$ als initiale Hyper-Resolutionsschritte möglich. In beiden Fällen läßt sich dies nicht in linearer Weise fortsetzen, da nur mit $\{\neg q, \neg r\}$ resolviert werden kann, was aber in jedem Fall ein negatives Literal einführt. Dagegen stellt $\langle\Gamma_1, \Gamma_2, \{\neg q, \neg r\}, \Box\rangle$ eine Hyper-Resolutionsableitung dar. ■

8.4.4 Stützmengenstrategie (SOS)

Schwache Konnektionstableaus waren ja eine beweiskonfluente „Approximation" der Konnektionstableaus, da sie durch die schwache Konnektionsbedingung immerhin eine gewisse Zielgerichtetheit aufrecht erhalten.

Gesucht ist also eine beweiskonfluente Approximation der linearen Resolution.

Die Idee dafür ist die Beobachtung, daß nicht beide Elternklauseln einer Resolvente aus derselben erfüllbaren Teilmenge der abgeleiteten Klauseln stammen sollten. Mindestens eine Elternklausel in jedem Resolutionsschritt sollte zur Unerfüllbarkeit beitragen.

Definition 8.69 *Sei* S *eine unerfüllbare Klauselmenge und* S' *eine erfüllbare Teilmenge. Dann heißt* $T = S - S'$ Stützmenge[9] *von* S. ∎

Definition 8.70 *Sei* T *Stützmenge einer unerfüllbaren Klauselmenge* S. *Der Resolutionsoperator* $\mathcal{R}_{SOS}$ *ist wie folgt definiert:*

$$\begin{array}{rcl} \mathcal{R}^0_{SOS}(S) & = & S \\ SOS^0(S) & = & T \\ \mathcal{R}^{i+1}_{SOS}(S) & = & \mathcal{R}^i_{SOS}(S) \cup \\ & & \{C \in \mathcal{R}(\mathcal{R}^i_{SOS}(S)) \mid \textit{mind. eine Elternklausel ist in } SOS^i(S)\} \\ SOS^{i+1}(S) & = & SOS^i(S) \cup (\mathcal{R}^{i+1}_{SOS}(S) - \mathcal{R}^i_{SOS}(S)) \\ \mathcal{R}^*_{SOS}(S) & = & \bigcup_{i=0}^{\infty} \mathcal{R}^i_{SOS}(S) \end{array}$$

∎

Falls man eine Behauptung der Form

$$(\bigwedge Ax_i) \to Thm$$

vorliegen hat, wobei ein Theorem aus einer Menge von Axiomen folgt, dann wird S von der Gestalt

$$\{Ax_1, \ldots, Ax_n, \neg Thm\}$$

sein. In diesem Fall sollten die Klauseln der SNF von „$\neg$Thm“ in der Stützmenge sein (wenn die Menge der Axiome nicht schon unerfüllbar ist).

Oft hat man nicht minimal unerfüllbare Formulierungen (vgl. das Beispiel mit der Modellierung des Zwischenwertsatzes); dann ist SOS unverzichtbar, weil dadurch eine gewisse Auswahl der verwendbaren Axiome stattfindet.

Implementierung von SOS Für die Implementierung wird die Menge der abgeleiteten Klauseln in zwei Listen aufgeteilt:

In `T` stehen diejenigen Klauseln, die den jeweiligen Support bilden, in `U` stehen alle übrigen Klauseln (*usable clauses*).

In jedem Schleifendurchlauf werden jeweils die Resolventen `R` einer gestützten Klausel `C` $\in$ `T` mit den übrigen Klauseln `U` $\cup$ `T` berechnet.

`C` wandert dadurch von `T` nach `U`, so daß die mehrfache Berechnung von Resolventen verhindert wird.

[9] Engl.: set of support (SOS).

```
/* S ist unerfüllbar, S - [SOS] ist erfüllbar */

R := ∅;
U := S - SOS;  % benutzbare Klauseln
T := SOS;      % aktuelle Stützmenge
while □ ∉ R and T ≠ ∅ do begin
  C := [select_clause(T)];  % faire Klauselauswahl
  T := delete(C,T);
  U := append(U,[C]);
  R := {E | E [Resolvente] von C und D ∈ U∪T, E ∉ U∪T};
  [deletion_strategy(R,U,T)];
  T := append(T,R)
end;
```

An den schattierten Stellen ist die Prozedur parametrisierbar:

SOS `U` muß erfüllbar sein, sonst ist die Wahl von SOS beliebig;

`select_clause` ist eine beliebige faire Auswahlstrategie;

Resolvente hier kann eine beliebige, mit SOS kompatible Resolutionsverfeinerung stehen;

deletion Löschregeln auf `T` und `U` getrennt oder zusammen angewandt.

Satz 8.71 *Sei* S *eine unerfüllbare PL-Klauselmenge mit Stützmenge* T. *Dann ist* $\square \in \mathcal{R}^*_{SOS}(S)$.

Beweis: 1. Weg: Sei $C \in T$, und sei S' eine minimal unerfüllbare Teilmenge von S, so daß $S' \supseteq ((S - T) \cup \{C\})$. Diese existiert, weil $S - T$ erfüllbar ist.

Nach Satz 8.32 gibt es eine lineare Resolutionsableitung von $\square$ aus S' mit Startklausel $C \in T$. Diese muß auch eine SOS-Ableitung sein, da jede Verbindungsklausel gestützt wird.

2. Weg: Sei I ein beliebiges Modell von $S - T$. Nach Satz 8.64 gibt es eine I-Clash-Resolutionsableitung von $\square$ aus S. Für jeden Resolutionsschritt gilt also, daß das Elektron in I fehlschlägt und daher nicht aus $S - T$ sein kann. Also hat jede Resolvente Support und wir haben auch eine SOS-Ableitung. ■

Beispiel 8.72 Die SOS Verfeinerung und damit die semantische Resolution überhaupt ist inkompatibel mit Lockresolution:

Mit den indizierten Klauselmengen $S = \{\{\overset{1}{p}\}, \{\overset{2}{\neg p}, \overset{3}{\neg q}\}, \{\overset{4}{q}\}\}$, $SOS = \{\{\overset{4}{q}\}\}$ ist kein SOS-Lockresolutionsschritt möglich, da $\{\overset{4}{q}\}$ Startklausel sein muß und $\neg q$ nirgends minimal vorkommt. Eine SOS-Widerlegung mit unlock(S) und unlock(SOS) ist trivial. ■

Semantische Resolution ist kompatibel mit den Löschregeln.

8.4.5 Geordnete Resolution mit Selektionsfunktion

In diesem Abschnitt betrachten wir eine Verfeinerung, welche Aspekte der geordneten und von Hyper-Resolution vereint. Ihre Definition ist relativ aufwendig, doch werden wir sehen, daß man etliche andere Verfeinerungen darunter subsumieren kann und man damit deren Vollständigkeit einfach gezeigt hat. Wir betrachten hier nur den aussagenlogischen Fall; die Definitionen und Beweise können durch Lifting auf die PL erweitert werden.

Klauseln sind nun Vielfachmengen von Literalen. Man beachte, daß die im folgenden verwendeten Begriffe einer zulässigen Ordnung bzw. einer Selektionsfunktion von den früher eingeführten etwas abweichen.

Definition 8.73 *Eine **zulässige Ordnung** ist eine wohlfundierte, totale Ordnung auf Atomen, so daß* $\neg p > p$ *für alle* p.

Sei $<$ *eine zulässige Ordnung.* f *ist eine **Selektionsfunktion** falls* f *jeder Klausel eine (möglicherweise leere) Teilmenge ihrer* negativen *Literale zuweist, wobei* $f(C) \neq \emptyset$*, falls das* $<$*-maximale Literal von* C *negativ ist.*

Eine zulässige Ordnung $<$ *wird auf Vielfachmengen wie folgt erweitert: seien* $L_1 \cdots L_n$ *und* $L'_1 \cdots L'_m$ *die Literale von Klauseln* C, D *in absteigender Reihenfolge bzgl.* $<$*. Dann gilt* $C < D$ *gdw* $L_1 \cdots L_n$ *vor* $L'_1 \cdots L'_m$ *liegt in der von* $<$ *induzierten lexikographischen Ordnung.* ∎

Definition 8.74 *Sei* $<$ *eine zulässige Ordnung,* f *eine Selektionsfunktion und* $C_1 = C'_1 \cup \{\!\{p_1\}\!\}, \ldots, C_n = C'_n \cup \{\!\{p_n\}\!\}$, $D = D' \cup \{\!\{\neg p_1, \ldots, \neg p_n\}\!\}$ *Klauseln.* $E = C'_1 \cup \cdots \cup C'_n \cup D'$ *ist eine* geordnete Resolvente mit Selektionsfunktion *von* $C_1, \ldots, C_n$ *und* D*, falls*

1. $f(D) = \{\neg p_1, \ldots, \neg p_n\}$
2. $f(C_1) = \cdots = f(C_n) = \emptyset$
3. p_i $<$*-maximal in* C'_i *für alle* i

Sei $D = D' \cup \{\!\{p, \ldots, p\}\!\}$. D' *ist ein* geordneter Faktor *von* D*, falls*

1. p $<$*-maximal in* D'
2. $f(D) = \emptyset$

∎

Man beachte, daß die C_i sämtlich positiv sein müssen, falls f alle negativen Literale selektiert. Damit kann man geordnete Resolution mit Selektionsfunktion auch als Verallgemeinerung von N-Resolution auffassen.

Zu geordneten Klauselmengen kann man eine Interpretation wie folgt konstruieren:[10]

[10] Interpretationen werden wie schon früher mit der Menge ihrer auf W gesetzten Atome identifiziert.

Definition 8.75 *Sei $<$ zulässige Ordnung, S Klauselmenge. Wir konstruieren Interpretationen*

$$I_C = \bigcup_{D<C} E_D$$

wobei

1. $E_C = \{p\}$, *falls*
   (a) p *$<$-maximal in* C *ist,*
   (b) $I_C \not\models C$
   (c) $I_C \cup \{p\} \not\models C - \{\!\{p\}\!\}$

   (In diesem Fall heißt C ***produktiv****.)*
2. $E_C = \{\}$ *sonst.*

$I = \bigcup_{C \in S} E_C$ *heißt die von* S ***induzierte Interpretation****.* ■

Man beachte, daß für die minimale Klausel M von S gilt: $I_M = \{\}$.

Der Beweis der Vollständigkeit der geordneten Resolution mit Selektionsfunktion ist technisch recht aufwendig. Wir skizzieren daher den Beweis zunächst informell:

- Betrachte eine bzgl. geordneter Resolution *saturierte* Klauselmenge S.
- Gib eine Modellkonstruktion für S an, die immer funktioniert, wenn S nicht die leere Klausel enthält.
- Das Modell wird iterativ konstruiert, beginned mit dem leeren Modell (alles falsch).
- Mache zuerst die bzgl. der Ordnung kleineren Klauseln wahr.
- Welche Atome „wahr gemacht“ werden, wird durch die Ordnung gesteuert.
- Aber: Ein technisch aufwendiger Beweis dafür ist erforderlich, daß die konstruierte Interpretation tatsächlich Modell aller Klauseln ist.

Beispiel 8.76 Die Ordnung sei $\neg q > q > p > r$; und die saturierte Klauselmenge sei $\{\neg q, p, r\} > \{q\} > \{p, r\}$.

Durch die Konstruktion wird zuerst das maximale Literal p in der Resolventen wahr gemacht, dann das maximale Literal q in der noch unerfüllten Elternklausel.

Das maximale Literal $\neg q$ in der maximalen Klausel kann jetzt zwar nicht wahr gemacht werden, aber aus der Konstruktion folgt, daß eines der anderen Atome (hier p), das auch in der Resolventen vorkommt, schon wahr ist. ■

Wir führen den oben skizzierten Beweis nun folmal aus.

Definition 8.77 *Eine Grundklauselmenge* S *ist* ***saturiert*** *falls* $\mathcal{R}^*_<(S) = S$, *wobei* $\mathcal{R}^*_<$ *die mit geordneter Resolution mit Selektionsfunktion und geordneter Faktorisierung herleitbaren Klauseln meint.* ■

Im folgenden ist I jeweils die von S induzierte Interpretation.

Das folgende Lemma besagt, daß Klauseln, die einmal während der Konstruktion von I wahr gemacht werden, auch im folgenden wahr bleiben. In anderen Worten, die Menge der von I_C erfüllten Klauseln wächst monoton in C bezgl. $<$.

Lemma 8.78 *Seien* C, D *Klauseln,* $D \in S$ *und* $C \leq D$. *Ist* $\neg p \in C$ *und* $p \notin I_D$, *dann* $p \notin I$.

Daraus folgt: aus $I_D \models C$ *folgt* $I \models C$ *und* $I_{D'} \models C$ *für alle* $D' > D$.

Beweis: Wäre $p \in I$, müßte es, da $p \notin I_D$, von einer Klausel produziert werden, die bzgl. $<$ größer als D (und C) ist.

Daraus würde $p > \neg p$ folgen, was der Zulässigkeit von $<$ widerspricht. ■

Lemma 8.79 *Seien* $C, D \in S$ *Klauseln mit* $C \leq D$.

Gilt $I_D \models C$, *dann auch* $I_{D'} \models C$ *für alle* $D' > D$, *und* $I \models C$.

Beweis: Einfaches Korollar von Lemma 8.78. ■

Lemma 8.80 *Sei* S *saturiert bzgl. geordneter Resolution mit Selektionsfunktion (incl. Faktorisierung) und sei* $\Box \notin S$. *Dann gilt für alle* $C \in S$:

Wird p *von* C *produziert, dann* $I \not\models C - \{\!\{p\}\!\}$

Beweis: Nach der Definition der Produktion gilt: $I_C \not\models C$ und also $I_C \not\models C' = C - \{\!\{p\}\!\}$.

Für beliebige (später wahr gemachte) $q \in I - I_C$ gilt: $q \geq p > C'$ (da p maximal in C). Also kommen solche q in C' nicht vor, und also $I \not\models C'$. ■

Das nächste Lemma besagt, daß in saturierten Klauselmengen jede Klausel C spätestens in der Konstruktion von E_C wahr gemacht wird. Die Beweisidee ist sehr ähnlich zum Vollständigkeitsbeweis für geordnete Tableaus: wird C nicht durch E_C wahrgemacht, ist sie nicht produktiv. Produktivitiät ist gerade so definiert, daß auf nicht produktive Klauseln eine der Inferenzregeln angewandt werden kann. Das Ergebnis dieser Anwendung muß natürlich in der saturierten Menge S enthalten sein, andererseits ist es kleiner als C, was zu einem Widerspruch führt.

Lemma 8.81 *Sei S saturiert bzgl. geordneter Resolution mit Selektionsfunktion (incl. Faktorisierung), und sei* $\square \notin S$. *Dann gilt für alle* $C \in S$*:*

1. *Wird* p *von* C *produziert, dann* $f(C) = \emptyset$.
2. $I_C \cup E_C \models C$.

Beweis: Widerspruchsbeweis. Sei C eine minimale Klausel, so daß die Behauptung *nicht* gilt.

Zu 1: Nehmen wir an, die 1. Bedingung sei verletzt. Es gebe selektierte Literale: $f(C) = \{\neg p_1, \ldots, \neg p_n\}$.

Es gilt $I_C \not\models \neg p_i$ (da $\neg p_i \in C$ und $I_C \not\models C$), also müssen die p_i von Klauseln $D_i = D_i' \cup \{\!\{p_i\}\!\} < C$ produziert worden sein, weshalb p_i maximal in D_i ist. Da C eine minimale Klausel ist, für die die Aussage des Lemmas nicht gilt, gilt sie für die D_i, also $f(D_i) = \emptyset$. Damit ist ein geordneter Resolutionsschritt mit Selektionsfunktion möglich mit Resolvente $R = D_1' \cup \cdots \cup D_n' \cup C''$, wobei $C'' = C - \{\!\{\neg p_1, \ldots, \neg p_n\}\!\}$.

Da S saturiert ist, gilt $R \in S$.

Aus $C > D_i$ folgt $C > D_i'$. Außerdem gilt trivialerweise $C > C''$; insgesamt also $C > R$.

Da C eine minimale Klausel ist, für die die Aussage des Lemmas nicht gilt, gilt sie für R, also $I_R \cup E_R \models R$. Und da $C > R$ folgt (Lemma 8.79): $I_C \models R$.

Aus Lemma 8.80 folgt, daß I und also I_C keines der D_i' erfüllt. (NB: man weiß nichts darüber, ob die D_i' in I_R erfüllt sind.)

Da I_C die D_i' nicht erfüllt, wohl aber R, folgt $I_C \models C''$ und also $I_C \models C$. Dies steht im Widerspruch zur Produktivität von C.

Zu 2: Nehmen wir an, die 2. Bedingung sei verletzt. Es gelte also $I_C \cup E_C \not\models C$.

Dann kann C nicht produktiv sein und muß daher eine der Bedingungen an produktive Klauseln verletzen. $I_C \not\models C$ ist gegeben, also ist entweder das maximale Literal in C negativ oder es gilt
$I_C \cup \{p\} \models C' = C - \{\!\{p\}\!\}$.

Behandeln wir zuerst den letzteren Fall; er kann nur eintreten (da $I_C \not\models C$), falls C' ein weiteres Auftauchen von p enthält: $C = C'' \cup \{\!\{p, p\}\!\}$. Da p maximal ist, und (wie in Fall 1) $f(C) = \emptyset$, ist durch Faktorisierung die kleinere Klausel C' ableitbar, für die ebenfalls
$I_{C'} \cup E_{C'} \not\models C'$ gilt (denn daraus würde $I_C \models C'$ und also $I_C \models C$ folgen). Also gilt auch für C' die Aussage des Lemmas nicht, was der Minimalität von C widerspricht.

Es bleibt der Fall zu betrachten, daß das maximale Literal in C negativ ist. Dann folgt aus der Zulässigkeit der Ordnung $>$, daß $f(C) \neq \emptyset$, also $f(C) = \{\neg p_1, \ldots, \neg p_n\}$. Wie im ersten Teil des Beweises folgt daraus ein Widerspruch (weil dann ein Resolutionsschritt möglich sein muß). ■

Satz 8.82 *Eine bzgl. geordneter Resolution mit Selektionsfunktion (und Faktorisierung) saturierte Klauselmenge ist genau dann unerfüllbar, wenn sie die leere Klausel enthält.*

Beweis: Enthält S die leere Klausel ist S trivialerweise unerfüllbar.

Umgekehrt enthalte S die leere Klausel nicht. Dann gilt nach Lemma 8.81: $I_C \cup E_C \models C$ für alle $C \in S$, woraus mit Lemma 8.79 $I \models C$ für alle $C \in S$ folgt; also gilt $I \models S$. ■

Der Kalkül kann ohne weiteres auf die erste Stufe gehoben werden, wofür man das am Ende von Abschnitt 7.2 erwähnte verallgemeinerte Lifting-Lemma benötigt.

Geordnete Resolution mit Selektionsfunktion ist eine sehr starke Verfeinerung. Man kann die Vollständigkeit der meisten anderen Verfeinerungen von Resolution darauf zurückführen, siehe Bachmair and Ganzinger [1997]. Als Beispiel betrachten wir die Vollständigkeit der N-Resolution und (damit) der allgemeinen semantischen Resolution.

Satz 8.83 *Sei die Ordnung* $>$ *beliebig, und selektiere* f *jeweils* alle *negativen Literale einer Klausel.*

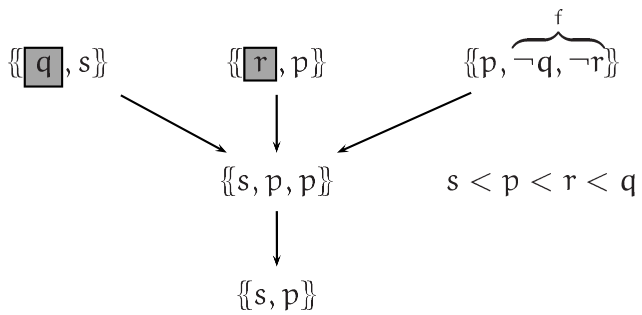
Dann ist jeder geordneter Resolutionsschritt mit Selektionsfunktion f *und anschließender Faktorisierung ein* N*-Resolutionsschritt.*

Beweis: Die Elektronen des geordneten Resolutionsschritts müssen positiv sein, sonst wäre in ihnen etwas selektiert, was aber verboten ist.

Im Kern sind genau die negativen selektiert, und diese werden gerade wegresolviert. ■

Gleiches kann man für Faktorisierungsschritte zeigen. Damit folgt aus der Vollständigkeit der geordneten Resolution mit Selektionsfunktion die Vollständigkeit der N-Resolution.

Beispiel 8.84 Der unten dargestellte N-Resolutionsschritt ist zugleich ein geordneter Resolutionsschritt.



Bei semantischer Resolution werden keine Mehrfachmengen verwendet. Darum ist die Verdoppelung des Atoms p kein Problem. ■

Die Vollständigkeit der (allgemeinen) semantischen Resolution kann aus der Vollständigkeit der N-Resolution gefolgert werden.

Definition 8.85 *Sei* S *eine Grundklauselmenge über* Σ*, und sei* I *eine* Σ*-Herbrand-Interpretation.*

Die von I induzierte Umbenennung σ_{I} *ersetzt (simultan) Literale wie folgt: wenn* $\mathrm{I} \models \mathrm{p}$ *wird jedes Auftauchen des Literals* p *durch* $\neg\mathrm{p}$ *und jedes Auftauchen des Literals* $\neg\mathrm{p}$ *durch* p *ersetzt; andernfalls bleiben* p *und* $\neg\mathrm{p}$ *unverändert.* ■

Beispiel 8.86 Sei $S = \{\{\neg\mathrm{p},\mathrm{q}\}, \{\mathrm{p},\mathrm{q},\neg\mathrm{r}\}, \{\neg\mathrm{s},\mathrm{t}\}, \{\neg\mathrm{s},\neg\mathrm{t}\}, \{\mathrm{r},\mathrm{p}\}, \{\neg\mathrm{q},\mathrm{s}\}\}$ über des Signatur $\Sigma = \{\mathrm{p},\mathrm{q},\mathrm{r},\mathrm{s},\mathrm{t}\}$; und sei $\mathrm{I} = \{\mathrm{q}\}$.

Dann ist $\sigma_{\mathrm{I}}(\mathrm{S}) = \{\{\neg\mathrm{p},\neg\mathrm{q}\}, \{\mathrm{p},\neg\mathrm{q},\neg\mathrm{r}\}, \{\neg\mathrm{s},\mathrm{t}\}, \{\neg\mathrm{s},\neg\mathrm{t}\}, \{\mathrm{r},\mathrm{p}\}, \{\mathrm{q},\mathrm{s}\}\}$. ■

Satz 8.87 *Sei* S *eine Grundklauselmenge über* Σ*, und sei* I *eine beliebige* Σ*-Herbrand-Interpretation. Ferner sei* B *ein N-Resolutionsbeweis für* $\sigma_{\mathrm{I}}(\mathrm{S})$*.*

Dann ist $\sigma_{\mathrm{I}}{}^{-1}(\mathrm{B})$ *ein semantischer Resolutionsbeweis für* S *bzgl. der Interpretation* I*.*

Beweisskizze: Der Beweis ist einfach und basiert darauf, daß $\mathrm{I}_{\mathrm{N}} \models \sigma_{\mathrm{I}}(\mathrm{p})$ gdw $\mathrm{I} \models \mathrm{p}$. ■

Aus dem Satz folgt, daß semantische Resolution bzgl. verschiedener Interpretationen gleichwertig ist, und insbesondere die Vollständigkeit bzgl. einer bestimmten Interpretation die Vollständigkeit bzgl. aller Interpretationen impliziert.

Historische Bemerkung: Das Konzept eines Clashs stammt von Robinson [1967].

Semantische (I-Clash-)Resolution (mit der zusätzlichen Ordnungseinschränkung) wurde von Slagle [1967] eingeführt.

UR-Resolution wurde von McCharen *et al.* [1976] eingeführt.

Hyper-Resolution und SOS sind mit die ersten Verfeinerungen gewesen und wurden von Robinson [1965a] (Hyper-Resolution) und von Wos *et al.* [1965] (SOS) definiert.

Geordnete Resolution mit Selektionsfunktion sowie die Modellkonstruktion aus saturierten Mengen stammt von Bachmair and Ganzinger [1994, 1997].

Kapitel 9

Gleichheit und andere Theorien

Theorieschließen ist eine wichtige Technik zur Steigerung der Effizienz im Automatischen Beweisen. Das Wissen aus oder über ein spezielles Gebiet (eine Theorie) wird genutzt, indem spezielle Methoden zum logischen Schließen in diesem Gebiet eingesetzt werden. Der allgemeine *Vordergrundbeweiser* ruft dazu einen speziellen *Hintergrundbeweiser* auf, um Probleme aus dem Spezialgebiet zu behandeln.

Der Einsatz von Theorieschließen ist für das Automatische Beweisen in vielen Anwendungsbereichen unverzichtbar. Insbesondere das Gleichheitsschließen ist äußerst wichtig, aber die meisten Modellierungen benutzen auch anderen Theorien, wie beispielsweise algebraische Theorien in mathematischen Problemen und Spezifikationen abstrakter Datentypen in Softwareverifikations-Problemen.

Wir legen fest, daß jede erfüllbare Menge geschlossener PL-Formeln eine Theorie ist:

Definition 9.1 *Eine* Theorie $\mathcal{T}$ *ist eine erfüllbare Menge geschlossener Formeln.* ■

In der Literatur wir oft die zusätzliche Forderung gestellt (neben der Erfüllbarkeit), daß eine Theorie unter der Folgerbarkeitsrelation geschlossen ist. Ohne diese Einschränkung müssen wir nicht zwischen einer Theorie und den sie definierenden Axiome unterscheiden.

Es folgen in diesem Abschnitt allgemeine Überlegungen zu

- den notwendigen Eigenschaften einer Theorie,
- der Schnittstelle zwischen Vordergrund- und Hintergrundbeweiser,
- Kriterien für die Korrektheit des jeweiligen Verfahrens

Allgemeine Kriterien zur Vollständigkeit sind schwierig aufzustellen. Im wesentlichen muß die Vollständigkeit jeweils neu für den einen neuen Hintergrundbeweiser bewiesen werden.

Die wichtigste Theorie in der Praxis ist die *Gleichheitstheorie*. Ihre Axiome drücken zwar nicht die Gleichheit von Elementen der Domäne im Modell aus, sondern nur deren Kongruenz; das genügt aber.

Beispiel 9.2 Die Gleichheits-Theorie $\mathcal{E}$ besteht aus folgenden Axiomen:

(1) $(\forall x)(x \approx x)$ (Reflexivität),

(2) Für alle Funktionssymbole in Σ:

$$\begin{aligned}&(\forall x_1)\cdots(\forall x_n)(\forall y_1)\cdots(\forall y_n)\\ &\qquad((x_1 \approx y_1 \wedge \ldots \wedge x_n \approx y_n) \to\\ &\qquad\; f(x_1,\ldots,x_n) \approx f(y_1,\ldots,y_n))\end{aligned}$$

(Montonie für Funktionssymbole),

(3) Für alle Prädikatensymbole in Σ:

$$\begin{aligned}&(\forall x_1)\cdots(\forall x_n)(\forall y_1)\cdots(\forall y_n)\\ &\qquad((x_1 \approx y_1 \wedge \ldots \wedge x_n \approx y_n) \to\\ &\qquad\;(p(x_1,\ldots,x_n) \to p(y_1,\ldots,y_n)))\end{aligned}$$

(Montonie für Prädikatensymbole),

Symmetrie und Transitivität Folgen aus Reflexivität und Monotonie für Prädikatensymbole. ■

Beispiel 9.3 Die Theorie $\mathcal{OP}$ der partiellen Ordnungen besteht aus den Axiomen

(1) $(\forall x)\neg(x < x)$ (Anti-Reflexivität),

(2) $(\forall x)(\forall y)(\forall z)((x < y) \wedge (y < z) \to (x < z))$
(Transitivität).

$\mathcal{OP}$ ist im Gegensatz zur Gleichheits-Theorie *endlich.* ■

Eine wichtige Klasse von Theorien sind die Erweiterungen der Gleichheits-Theorie.

Beispiel 9.4 Die AC-Theorie für das Funktionssymbol f besteht (neben den $\mathcal{E}$-Axiomen) aus den zusätzlichen Axiomen

$$(\forall x)(\forall y)(\forall z)(f(f(x,y),z) \approx f(x,f(y,z)))$$

und

$$(\forall x)(\forall y)(f(x,y) \approx f(y,x)) \ .$$

■

Weitere typische Beispiele für Erweiterungen der Gleichheits-Theorie sind die Spezifikationen algebraischer Strukturen wie beispielsweise Gruppen.

Die Definitionen von Interpretation, Erfüllbarkeit, Tautologie etc. werden an das Theorieschließen angepaßt:

Definition 9.5 *Sei $\mathcal{T}$ eine Theorie.*

Eine $\mathcal{T}$-interpretation *ist eine Interpretation, die alle Formeln in $\mathcal{T}$ erfüllt.*

Eine Formel ϕ (eine Menge Φ von Formeln) ist $\mathcal{T}$-erfüllbar, *wenn es eine $\mathcal{T}$-Interpretation gibt, die sie erfüllt. Andernfalls ist sie $\mathcal{T}$-unerfüllbar.*

Eine geschlossene Formel ist $\mathcal{T}$-allgemeingültig, *wenn sie von allen $\mathcal{T}$-interpretationen erfüllt wird.*

Eine Formel ϕ ist eine logische $\mathcal{T}$-Konsequenz *einer Menge Ψ von Formeln, in Zeichen $\Psi \models_{\mathcal{T}} \phi$, wenn ϕ von allen $\mathcal{T}$-Interpretationen erfüllt wird, die Ψ erfüllen.*

Eine Formel ϕ ist eine starke $\mathcal{T}$-Konsequenz *einer Menge Ψ von Formeln, in Zeichen $\Psi \models^{\circ}_{\mathcal{T}} \phi$, falls für alle $\mathcal{T}$-interpretationen $\langle \mathrm{D}, \mathrm{I} \rangle$ und alle Variablenbelegungen ν:*

$$\textit{Wenn } \mathit{val}_{\mathrm{I},\nu}(\psi) = \mathsf{T} \textit{ für alle } \psi \in \Psi, \textit{ dann } \mathit{val}_{\mathrm{I},\nu}(\phi) = \mathsf{T}.$$

■

Proposition 9.6 *Gegeben eine Theorie $\mathcal{T}$, eine Menge Φ geschlossenere Formeln und eine geschlossene Formeln ψ; dann sind die folgenden Aussagen äquivalent:*

1. $\Phi \models_{\mathcal{T}} \psi$.
2. $\Phi \cup \mathcal{T} \models \psi$.
3. $\Phi \cup \mathcal{T} \cup \{\neg\psi\}$ *ist unerfüllbar.*
4. $\Phi \cup \{\neg\psi\}$ *ist $\mathcal{T}$-unerfüllbar.*

9.1 Eigenschaften von Theorien

Definition 9.7 *Eine Theorie $\mathcal{T}$ ist* (endlich) axiomatisierbar, *falls es eine (endliche)* entscheidbare *Menge Ψ geschlossener Formeln gibt (die Axiome), so daß:*

$$\phi \textit{ ist } \mathcal{T}\textit{-allgemeingültig gdw. } \Psi \models \phi.$$

Eine Theorie $\mathcal{T}$ ist vollständig, *falls für alle geschlossenen Formeln ϕ entweder ϕ oder $\neg\phi$ $\mathcal{T}$-allgemeingültig ist.* ■

Nur axiomatisierbare Theorien können im weiteren behandelt werden. Ein Beispiel für eine Theorie die nicht axiomatisierbar ist, ist die Menge aller erfüllbaren Formeln.

Die Menge der $\mathcal{T}$-allgemeingültigen Formeln einer aximatisierbaren Theorie $\mathcal{T}$ ist aufzählbar – im allgemeinen aber unentscheidbar (Beispiel: leere Theorie). Die Menge der $\mathcal{T}$-allgemeingültigen Formeln einer aximatisierbaren *und* vollständigen Theorie $\mathcal{T}$ ist entscheidbar.

Eine Alternative zur Axiomatisierung ist die Definition einer Theorie als Menge aller Formeln, die in einer bestimmten Interpretation wahr sind. Solche Theorien sind immer vollständig. Für das Automatische Beweisen ist die Technik aber ungeeignet.

Definition 9.8 *Eine Theorie $\mathcal{T}$ ist* universell*, falls sie mit Hilfe einer Menge von Axiomen axiomatisierbar ist, die in Skolem-Normalform sind.* ∎

Für universelle Theorien gilt folgende Variante des Herband-Theorems:

Satz 9.9 *Eine Menge Φ von Formeln in SNF ist genau dann $\mathcal{T}$-unerfüllbar, wenn es eine endliche Menge von Grundinstanzen von Φ gibt, die $\mathcal{T}$-unerfüllbar ist.*

Beispiel 9.10 Die Theorie $\mathcal{T} = \{(\exists x)p(x)\}$ ist *nicht* universell.

Die Menge $\Phi = \{(\forall x)(\neg p(x))\}$ ist $\mathcal{T}$-unerfüllbar. Die Menge *aller* Grundinstanzen von Φ ist jedoch $\mathcal{T}$-erfüllbar; sie kann mit einer $\mathcal{T}$-Interpretation erfüllt werden, in der nicht alle Elemente des Universums durch Grundterme erreicht werden. ∎

Da das Herbrand-Theorem essentiell ist für das AB, werden alle weiteren Überlegungen auf universelle Theorien beschränkt. Dies ist keine allzu schwere Einschränkung, denn eine nicht-universelle Theorie kann durch Skolemisieren universell gemacht werden. Allerdings muß dann die besondere Bedeutung der eingeführten Skolem-Symbole vom Hintergrundbeweiser richtig gehandhabt werden.

Beispiel 9.11 Eine Erweiterung von $\mathcal{OP}$, die das Dichtheits-Axiom

$$(\forall x)(\forall y)((x < y) \rightarrow (\exists z)((x < z) \wedge (z < y)))$$

enthält, ist nicht universell.

Verwende statt dessen:

$$(\forall x)(\forall y)((x < y) \rightarrow ((x < f(x, y)) \wedge (f(x, y) < y)))$$

Ein Hintergrundbeweiser, der diese Theorie behandelt, muß nun wissen, daß stets $x < f(x, y)$ gilt, und beispielweise $f(a, b) < a$ einen Widerspruch darstellt.

∎

Key	$\mathcal{E}$-Refuter
$\{\neg(a \approx a)\}$	id
$\{\neg(x \approx a)\}$	$\{x \leftarrow a\}$
$\{(\forall x)(\neg(x \approx a))\}$	id
$\{p(a), \neg p(b)\}$	$\langle \mathrm{id}\quad, \{\neg(a \approx b)\}\rangle$
$\{p(f(a), f(b)), f(x) \approx x\}$	$\langle\{x \leftarrow a\}, \{p(a, f(b))\}\rangle$ $\langle\{x \leftarrow b\}, \{p(f(a), b)\}\rangle$
$\{p(f(a), f(b)), (\forall x)(f(x) \approx x)\}$	$\langle \mathrm{id}\quad, \{p(a, b)\}\rangle$

Tabelle 9.1: Beispiele für $\mathcal{E}$-Refuter.

9.2 Grundbegriffe des Theorieschließens

Man beachte, daß in der folgenden Definition die *starke* Konsequenzrelation verwendet wird.

Definition 9.12 *Sei Φ eine endliche Menge von Formeln – der* Key*.*

Eine endliche Menge $R = \{\rho_1, \ldots, \rho_k\}$ von Formeln, die leer sein kann, ist ein $\mathcal{T}$-Residuum von Φ, falls es eine (idempotente) Substitution σ gibt, so daß

1. $R = R\sigma$,
2. $\Phi\sigma \models^{\circ}_{\mathcal{T}} \rho_1 \vee \ldots \vee \rho_k$

*Dann heißt das Paar $\langle\sigma, R\rangle$ $\mathcal{T}$-*Refuter *von Φ.*

Ist das Residuum leer, wird σ mit $\langle\sigma, \emptyset\rangle$ identifiziert und Refuter genannt. ■

Die zweite Bedingung in der obigen Definition eines Residuums wird zu „alle Grundinstanzen von $\Phi\sigma$ sind $\mathcal{T}$-unerfüllbar“, wenn das Residuum leer ist.

Beispiel 9.13 Tabelle 9.1 zeigt einige Beispiele für $\mathcal{E}$-Refuter. ■

Definition 9.14 *Eine Menge Φ von Formeln ist $\mathcal{T}$-*komplementär*, falls für alle $\mathcal{T}$-Interpretationen $\mathcal{D} = \langle D, I\rangle$ und alle Variablenbelegungen ν gilt:*

$$val_{I,\nu}(\Phi) = F \ .$$

■

„$\mathcal{T}$-komlementär“ verallgemeinert den Begriff „komplementär“. $\mathcal{T}$-komplementäre Formeln können zum Abschluß eines Tableauastes verwendet werden. Die Intuition zu „komplementär“ ist, daß, auch wenn man später freie Variablen instantiiert, der Widerspruch erhalten bleiben.

Beispiel 9.15 Die Menge $\{\neg(x \approx y)\}$ ist $\mathcal{E}$-unerfüllbar; sie ist aber *nicht* $\mathcal{E}$-komplementär.

Die Menge $\{\neg(x \approx x)\}$ ist $\mathcal{E}$-unerfüllbar und $\mathcal{E}$-komplementär. ■

Lemma 9.16 *Eine Substitution* σ *und eine Menge* $R = \{\rho_1, \ldots, \rho_k\}$ *von Formeln bilden genau dann einen Refuter* $\langle \sigma, R \rangle$ *für einen Key* Φ, *wenn*

1. $R = R\sigma$;

2. $\Phi\sigma \cup \{\neg\rho_1, \ldots, \neg\rho_k\}$ $\mathcal{T}$*-komplementär ist.*

Satz 9.17 *Sei* $\mathcal{T}$ *universell, und gebe es unendlich viele Funktionssymbole jeder Stelligkeit, die nicht in* $\mathcal{T}$ *vorkommen.*

Dann ist eine quantorenfreie Formel Φ *genau dann* $\mathcal{T}$*-komplementär, wenn alle Grundinstanzen von* Φ $\mathcal{T}$*-unerfüllbar sind.*

Beispiel 9.18 Sei $\mathcal{T} = \{p(t) \mid t \text{ ein Grundterm}\}$. In dieser Theorie kommen alle Funktionssymbole vor (obiges Theorem ist also nicht anwendbar).

Und tatsächlich ist – entgegen der Aussage des Theorems – die Formel $\neg p(x)$ ist *nicht* $\mathcal{T}$-komplementär, obwohl alle ihre Grundinstanzen $\mathcal{T}$-unerfüllbar sind. ■

Definition 9.19 *Ein* Hintergrundbeweiser *für eine Theorie* $\mathcal{T}$ *ist (formal) eine partielle Funktion*

$$\mathcal{R} : 2^{Form_\Sigma} \longrightarrow 2^{Subst_\Sigma \times 2^{Form_\Sigma}} ,$$

so daß für alle Keys Φ, *für die* $\mathcal{R}$ *definiert ist,* $\mathcal{R}(\Phi)$ *eine Menge von* $\mathcal{T}$*-Refutern für* Φ *ist.*

Ein Hintergrundbeweiser ist total, *falls alle Residuen, die er liefert, leer sind; andernfalls ist er* partiell. ■

Beim totalen Theorieschließen überläßt man dem Hintergrundbeweiser das finden von Widersprüchen (Astabschlüssen) – und nur das.

Beim partiellen Schließen kann der Hintergrundbeweiser auch Zwischenergebnisse liefern. Es gilt dann: Zeigt der Vordergrundbeweiser, daß

$$(\Psi\sigma \cup \{\rho_1 \vee \ldots \vee \rho_k\})\tau$$

$\mathcal{T}$-unerfüllbar ist, dann beweist das die $\mathcal{T}$-Unerfüllbarkeit von $\Psi\sigma\tau$, denn

$$(\Psi\sigma \cup \{\neg(\rho_1 \vee \ldots \vee \rho_k)\})$$

ist $\mathcal{T}$-complementär, also

$$(\Psi\sigma \cup \{\neg(\rho_1 \vee \ldots \vee \rho_k)\tau\})$$

ebenfalls $\mathcal{T}$-unerfüllbar.

Beispiel 9.20 Ein Hintergrundbeweiser $\mathcal{R}$ für die Theorie partieller Ordnungen ($\mathcal{PO}$) kan wie folgt definiert werden: $\mathcal{R}(\Phi)$ ist die kleinste Menge, so daß

1. wenn $t < t'_1, t'_2 < t'' \in \Phi$ mit t'_1, t'_2 unifizierbar, dann $\langle \sigma, t\sigma < t''\sigma \rangle \in \mathcal{R}(\Phi)$, wobei σ ein mgu ist;

2. wenn $t_1 < t_2 \in \Phi$ mit t_1, t_2 unifizierbar, dann $\sigma \in \mathcal{R}(\Phi)$, wobei σ ein mgu ist.

Die Kombination dieses Hintergrundbeweisers mit FV-Tableaus liefert einen vollständigen Kalkül für die Theorie $\mathcal{PO}$. ■

Hintergrundbeweiser können ggf. mit Regelschemata beschrieben werden.

Beispiel 9.21 Der Hintergrundbeweiser für die Theorie partieller Ordnungen aus Beispiel 9.20 kann durch folgende Schemata beschrieben werden:

$$\frac{\begin{array}{c} t < t'_1 \\ t'_2 < t'' \end{array}}{t\sigma < t''\sigma} \qquad\qquad \frac{t_1 < t_2}{*}$$

$$\begin{array}{c} \sigma \text{ ein mgu} \\ \text{von } t'_1, t'_2 \end{array} \qquad\qquad \begin{array}{c} \sigma \text{ ein mgu} \\ \text{von } t_1, t_2 \end{array}$$

■

9.3 Theorieschließen in Grundterm-Tableaus

Um Grundterm-Tableaus um Theorieschließen zu erweitern, fügen wir zusätzliche zu den α-, β-, γ-, δ-Regeln und der Standard-Abschluß-Regel die folgenden Regeln hinzu:

$$\frac{\begin{array}{c} \phi_1 \\ \vdots \\ \phi_p \end{array}}{\rho_1 \mid \cdots \mid \rho_k} \qquad\qquad \frac{\begin{array}{c} \phi_1 \\ \vdots \\ \phi_p \end{array}}{*}$$

wobei $\langle id, \{\rho_1, \ldots, \rho_k\} \rangle$ $(k \geq 1)$ ein Refuter für den Key $\{\phi_1, \ldots, \phi_p\}$ ist, und $\rho_1, \ldots, \rho_k$ keine freien Variablen enthält.

wobei id ein Refuter für den Key $\{\phi_1, \ldots, \phi_p\}$ ist.

Zur Kombination mit Grundterm-Tableaus sind nur solche Hintergrundbeweiser geeignet, die Residuen ohne freie Variablen liefern; und die Substitution in den Refutern müssen immer leer sein.

9.4 Theorieschließen in Tableaus mit freien Variablen

Um FV-Tableaus um Theorieschließen zu erweitern, fügen wir zusätzliche zu den α-, β-, γ-, δ-Regeln und der Standard-Abschluß-Regel der FV-Tableaus die folgenden Regeln hinzu:

$$\frac{\begin{array}{c}\phi_1\\ \vdots\\ \phi_p\end{array}}{\rho_1 \mid \cdots \mid \rho_k}$$

wobei $\langle\sigma, \{\rho_1, \dots, \rho_k\}\rangle$ $(k \geq 1)$ ein Refuter für den Key $\{\phi_1, \dots, \phi_p\}$ ist, und σ auf das Tableau angewendet wird.

$$\frac{\begin{array}{c}\phi_1\\ \vdots\\ \phi_p\end{array}}{*}$$

wobei σ ein Refuter für den Key $\{\phi_1, \dots, \phi_p\}$ ist und auf das Tableau angewendet wird.

Eine Erweiterung von Theorieschließen auf Tableaus mit universellen Variablen ist durch Einschränkung der Substitution σ auf die nicht-universellen Variablen.

9.5 Theorieschließen in Resolution

Auch die Resolutionsregel kann leicht um Theorieschließen erweiert werden:

$$\frac{\begin{array}{c}\{\phi_1\} \cup C_1\\ \vdots\\ \{\phi_p\} \cup C_p\end{array}}{R \cup (C_1 \cup \ldots \cup C_p)\sigma}$$

wobei $\langle\sigma, R\rangle$ ein Refuter für den Key $\{\phi_1, \dots, \phi_p\}$ ist

Auch hier werden die Variablen in den Prämissen vor Anwendung der Regel umbenannt.

9.6 Korrektheit und Vollständigkeit von Theorieschließen

Die Korrektheit folgt direkt aus der Definition, nach der Hintergrundbeweiser nur korrekte Refuter liefern dürfen.

Vollständigkeit ist zumindest dann gegeben, wenn:

- der Vordergrundbeweiser durch Fairneß alle möglichen Keys generiert und diese auch an den Hintergrundbeweiser gibt,
- der Hintergrundbeweiser *alle* möglichen Refuter liefert,
- der Vordergrundbeweiser die vom Hintergrundbeweiser gelieferten Refuter in fairer Weise alle verwendet.

Die Vollständigkeit auf diese Weise sicherzustellen ist jedoch nicht praktikabel, weil der Suchraum viel zu groß ist, wenn alle möglichen Refuter verwendet werden. Man muß die Suche auf „allgemeinste" Refuter beschränken.

9.7 Allgemeinste Refuter

Definition 9.22 *Sei $\mathcal{T}$ eine Theorie, und sei W eine Menge von Variablen. Dann sind die Subsumtionsrelationen $\leq^W$ und $\leq^W_\mathcal{T}$ auf Refutern definiert durch:*

(a) $\langle \sigma, R \rangle \leq^W \langle \sigma', R' \rangle$, *falls es eine Substitution τ gibt, so daß*

1. $\sigma'(x) = \sigma(x)\tau$ *für alle* $x \in W$, *und*
2. $R'\tau \supset R$.

(b) $\langle \sigma, R \rangle \leq^W_\mathcal{T} \langle \sigma', R' \rangle$, *falls es einen Refuter* $\langle \sigma'', R'' \rangle$ *gibt, so daß*

1. $\langle \sigma, R \rangle \leq^W \langle \sigma'', R'' \rangle$, *und*
2. $\Phi\sigma'' \cup \{\bigvee R''\} \models^\circ_\mathcal{T} \Phi\sigma' \cup \{\bigvee R'\}$ *für alle Formelmengen Φ (insbesondere auch für $\Phi = \emptyset$).*

■

Die Menge W enthält die „relevanten" Variablen (d. h., zumindest die in den Refutern vorkommenden). Bei Tableaus sind es die im jeweiligen Tableau vorkommenden, bei Resolution die in den beteiligten Klauseln vorkommenden Variablen.

Man beachte, daß bzgl. der Relation $\leq^W$ kleinere Refuter allgemeiner sind, also ist ein minimaler Refuter ein allgemeinster.

Die intuitive Bedeutung von $\langle\sigma, R\rangle \leq_{\mathcal{T}}^{W} \langle\sigma', R'\rangle$ ist: Der Effekt des Refuters $\langle\sigma', R'\rangle$ kann simuliert werden, indem man zuerst die Substitution τ anwendet und dann den entstehenden Refuter $\langle\sigma'', R''\rangle$ verwendet, von dem der Refuter $\langle\sigma', R'\rangle$ eine logische Konsequenz ist.

Beispiel 9.23 Sei $\mathcal{T}$ die leere Theorie. $\langle id, \{p(x)\}\rangle$ subsumiert $\langle\{x \leftarrow a\}, \{p(a)\}\rangle$ für alle Variablenmengen W; $\langle id, \{p(x)\}\rangle$ subsumiert $\langle id, \{p(a)\}\rangle$
nur, falls $x \notin W$. ∎

Beispiel 9.24 Sei $\mathcal{T}$ beliebig. $\langle\sigma, \{\phi\}\rangle$ ist allgemeiner als $\langle\sigma, \{\phi, \psi\}\rangle$. ∎

Beispiel 9.25 Sei $\mathcal{T}$ die Gleichheitstheorie $\mathcal{E} \cup \{a \approx b\}$. Dann subsumieren sich $\langle id, p(a)\rangle$ und $\langle id, p(b)\rangle$
bzw. $\{x \leftarrow a\}$ und $\{x \leftarrow b\}$ gegenseitig bzgl. $\leq_{\mathcal{T}}$ ∎

Proposition 9.26 *Die Vollständigkeit des Gesamtkalküls bleibt erhalten, wenn nur bzgl. $\leq^{W}$ bzw. $\leq_{\mathcal{T}}^{W}$ minimale Refuter verwendet werden.*

Die Relation $\leq_{\mathcal{T}}^{W}$ ist „mächtiger“ als $\leq^{W}$ aber auch deutlich schwerer zu berechnen.

9.8 Partielles Gleichheits-Schließen (Paramodulation)

Folgende Regelschemata definieren einen vollständigen, korrekten Hintergrundbeweiser für die Gleichheitstheorie:

$$\frac{\begin{array}{c} t \approx s \\ \phi[t'] \end{array}}{(\phi[s])\sigma} \qquad \frac{\begin{array}{c} s \approx t \\ \phi[t'] \end{array}}{(\phi[s])\sigma} \qquad \frac{\neg(t \approx t')}{*}$$

wobei jeweils σ ein mgu von t und t' ist.

Gewissermaßen wenden diese Regeln Gleichungen an.

Der Begriff „Paramodulation“ stammt aus Resolutionswelt. Historisch wurden diese und die anderen Gleichheits-Regeln zunächst alle unabhaengig von Theorieschliessen formuliert (für Tableaus oder Resolution).

Beispiel 9.27 Abbildung 9.1 zeigt einen Tableaubeweis für die Unerfüllbarkeit der aus (1)–(5) bestehenden Formelmenge.

Bei Ableitung von (10) aus (4) mit (8) wird die Substitution $\sigma_1 = \{x_2 \leftarrow a\}$ angewendet; und bei der Ableitung von (11) aus (10) mit (6) wird die Substitution $\sigma_2 = \{x_1 \leftarrow a\}$ angewendet.

$$
\begin{array}{ll}
(1) & (\forall x)(g(x) \approx f(x) \vee \neg(x \approx a)) \\
(2) & (\forall x)(g(f(x)) \approx x) \\
(3) & b \approx c \\
(4) & p(g(g(a)), b) \\
(5) & \neg p(a, c) \\
(6) & g(f(x_1)) \approx x_1 \\
(7) & g(x_2) \approx f(x_2) \vee \neg(x_2 \approx a) \\
(8) & g(x_2) \approx f(x_2) \qquad (9) \quad \neg(x_2 \approx a) \\
(10) & \boxed{p(g(f(a)), b)} \\
(11) & \boxed{p(a, b)} \\
(12) & \boxed{p(a, c)}
\end{array}
$$

Abb. 9.1: Ein Tableaubeweis, der die Paramodulationsregeln verwendet (Beispiel 9.27).

Wird Gleichung (8) auf (4) in falscher Weise angwendet, wird nämlich (10') $p(f(g(a)), b)$ statt (10) $p(g(f(a)), b)$ abgeleitet, dann wird der Term $g(a)$ für x_2 substituiert und das Tableau kann nicht geschlossen werden. Dann braucht man eine weitere Instanz von (7), (8) und (9) oder aber man muß backtracken. ■

9.9 Partielles Gleichheits-Schließen (RUE-Resolution)

Ein weiterer korrekter Hintergrundbeweiser für die Gleichheitstheorie kann durch folgende Regelschemata definiert werden:

$$
\frac{\begin{array}{c} p(t_1, \dots, t_k) \\ \neg p(s_1, \dots, s_k) \end{array}}{\neg(t_1 \approx s_1) \mid \cdots \mid \neg(t_k \approx s_k)}
\qquad
\frac{\neg(f(t_1, \dots, t_k) \approx f(s_1, \dots, s_k))}{\neg(t_1 \approx s_1) \mid \cdots \mid \neg(t_k \approx s_k)}
$$

$$
\frac{t \approx s}{s \approx t} \qquad \frac{\neg(t \approx t')}{*}
$$

Bei den ersten drei dieser Regeln ist die Substitution σ im Refuter leer, bei der vierten Regel ist sie ein mgu von t und t'.

Der Nachteil der RUE-Regeln ist ihr hoher Verzweigungsgrad.

9.10 Totales Gleichheits-Schließen

Ein vollständiger, totaler Hintergrundbeweiser für Gleichheit muß die folgenden Refuter liefern – dabei bezeichne $E(\Phi)$ die Menge der (nicht-quantifizierten) Gleichungen $l \approx r$ in einem Key Φ:

Eine Substitution σ ist ein Refuter für Φ, wenn

$$(E(\Phi))\sigma \models^{\circ}_{\mathcal{E}} \langle t_1, \dots, t_n \rangle\sigma \approx \langle s_1, \dots, s_n \rangle\sigma$$

für ein Paar $p(t_1, \dots, t_n)$ und $\neg p(s_1, \dots, s_n)$ in Φ, oder

$$(E(\Phi)\sigma) \models^{\circ}_{\mathcal{E}} t\sigma \approx s\sigma$$

für $\neg(t \approx s)$ in Φ.

Um solche Refuter zu berechnen, müssen *starre* E-*Unifikationsprobleme* gelöst werden (diese heißen starr, weil die Variablen in den Gleichungen nicht universell quantifiziert sind).

Definition 9.28 *Ein* starres E-Unifikationsproblem $\langle E, s, t \rangle$ *besteht aus*

1. *einer Menge* E *von (quantorenfreien) Gleichungen*
2. *Termen* s *und* t

Eine Substitution σ *ist eine Lösung von* $\langle E, s, t \rangle$*, wenn*

$$E\sigma \models^{\circ}_{\mathcal{E}} s\sigma \approx t\sigma$$

■

Eine Substitution, die ein aus einem Tableau-Ast extrahiertes starres E-Unifikationsproblem löst, schließt diesen Ast.

Zum Test, ob alle Äste eines Tableaus zugleich geschlossen werden können, müssen *simultane* starre E-Unifikationsprobleme gelöst werden:

Definition 9.29 *Ein* simultanes *starres* E-*Unifikationsproblem ist eine endliche Menge starrer* E-*Unifikationsprobleme.*

Eine Substitution ist Lösung eines simultanen Problems, wenn sie Lösung jedes seiner Teilprobleme ist. ■

Beispiel 9.30 Wir betrachten wieder das Beispieltableau aus Abbildung 9.1 – allerdings ohne die eingerahmten, durch Paramodulation hinzugefügten Formeln. Bezeichne B_1 den linker Ast des Tableaus und B_2 den rechten Ast.

Die Glechungsmengen sind

$$\begin{aligned} E(B_2) &= \{\, b \approx c,\ g(f(x_1)) \approx x_1 \,\} \\ E(B_1) &= E(B_2) \cup \{\, g(x_2) \approx f(x_2) \,\} \end{aligned}$$

Die aus B_2 zu extrahierenden starren E-Unifikationsprobleme sind:

$$\begin{array}{ll} P_1: & \langle E(\Phi_2), \langle g(g(a)), b\rangle, \langle a, c\rangle\rangle \\ P_2: & \langle E(\Phi_2), x_2, a\rangle \end{array}$$

Das aus B_1 Ast zu extrahierende starre E-Unifikationsproblem ist:

$$P_3: \ \langle E(\Phi_1), \langle g(g(a)), b\rangle, \langle a, c\rangle\rangle$$

Die Substitution $\{x_1 \leftarrow a,\, x_2 \leftarrow a\}$ ist eine simultane Lösung von P_2 und P_3, woraus folgt, daß sie erlaubt, das gesamte Tableau abzuschließen. ■

Universelle E-Unifikation (also E-Unifikation mit universell quantifizierten Gleichungen) ist unentscheidbar. Dagegen ist die starre E-Unifikation (überraschenderweise) entscheidbar; die Lösbarkeit starrer E-Unifkationsprobeleme ist ein NP-vollständiges Problem. Simultane starre E-Unifikation ist jedoch (wiederum überraschend) unentscheidbar.

In der Literatur sind einige (falsche) Beweise für die falsche Behauptung zu finden, simultane starrer E-Unifikation sei entscheidbar. Der Beweis der Unentscheidbarkeit gelang schließlich durch Reduktion der als unentscheidbar bekannten Unifikation höherer Ordnung auf simultane starre E-Unifikation.

Es ist möglich, eine minimale Lösung zu einem starren E-Unifikationsproblem effizient zu berechnen. Es ist aber *nicht* möglich eine Lösung für eine simultanes Problem durch Kombination der minimalen Lösungen seiner Teilprobleme zu berechnen.

Ein vollständiger Tableaukalkül entsteht, wenn man *alle* Lösungen der aus den Ästen extrahierten Probleme aufzählt und (fair) verwendet (nicht nur die minimalen).

Folgende Frage ist ein offenes Forschungsproblem: Kann ein vollständiger Tableaukalkül konstruiert werden, der nur minimale (oder jedenfalls nur endlich viele) Lösungen der extrahierten Probleme verwendet? Ein solcher Kalkül wäre besonders effizient.

Kapitel 10

Evaluierung von Theorembeweisern

10.1 Vergleichsmöglichkeiten

Wir listen zunächst eine Reihe möglicher Vergleichsparameter auf. Es ist auch jeweils angegeben für welche Formelklassen der Vergleich derzeit zur Verfügung steht. Man beachte, daß sich die meisten Ansätze auf deterministische Suchprozeduren beziehen, nicht auf die indeterministischen Kalkülregeln.

Theoretische Vergleiche

- Die Länge des kürzesten möglichen Beweises bezüglich eines realistischen Begriffs davon, was ein elementarer Beweisschritt sei (AL + PL).
- Die absolute Größe des Suchraums (nur AL-Horn).
- Berechnung des Erwartungswertes der durchschnittlichen Beweislänge für zufallsgenerierte Formelklassen in Abhängigkeit verschiedener Parameter (z.B.: das Verhältnis der Anzahl der Klauseln zur Größe der Signatur) (nur AL).
- Qualitative Aussagen wie: Beweiskonfluenz

Praktische Vergleiche

- Laufzeitvergleiche implementierter Prozeduren für Benchmark-Suiten.
- Fallstudien in Anwendungen.
  - reine Mathematik, z.B. Algebra, Logik, Kombinatorik
  - formale Hard- und Softwareverifikation

10.2 Relative Beweiskomplexität von Kalkülen

Die Länge des kürzesten Beweises entspricht der indeterministischen Stärke von Kalkülen.

Die Funktion $|\ldots|$ ist auf diversen Beweisprozeduren und auf Klauselmengen definiert. Für Klauseltableaus liefert sie die Anzahl der Knoten in einem Tableau; für Resolutionsableitungen die Länge der Ableitung; für Klauselmengen schließlich liefert $|\ldots|$ die Anzahl der *Auftauchen* von Literalen.

Die Funktion $|\ldots|$ sollte „realistisch" sein: sie sollte die tatsächlichen Kosten in Termini von elementaren Maschinenschritten für die Konstruktion eines Beweises modulo eines polynomiellen Faktors erkennen lassen. Problematisch wäre es zum Beispiel wenn in den Literalen eines Beweises nach polynomiell vielen Schritten Terme exponentieller Länge auftauchen könnten.

Dies könnte durch eine unbeschränkte Schnittregel verursacht werden:

$$\frac{}{p(t_1, \ldots, t_m) \mid \neg p(t_1, \ldots, t_m)} \quad p(t_1, \ldots, t_m) \in At_\Sigma$$

Man kann zeigen, daß für alle bisher betrachteten Klauseltableauvarianten die Knotenzahl ein in diesem Sinne realistisches Maß ist.

Das Zählerbeispiel (Beispiel 7.16) mit $S = \{\{p(0)\}, \{\neg p(x), p(s(x))\}\}$ zeigt, daß in PL-Resolutionsableitungen zwischen der Anzahl elementarer Maschinenschritte und der Anzahl der Ableitungsschritte kein polynomieller Zusammenhang besteht, weil in polynomiell vielen Schritten Substitutionen exponentieller Länge erzeugt werden können.

Wir nannten dies „realistisch", Letz [1993] nennt diese Eigenschaft „polynomielle Transparenz".

Allerdings ist S erfüllbar, und das Hinzufügen von $\{\neg p(s^{2^n}(0))\}$ führt einen exponentiellen Term in S ein. Das Beispiel kann jedoch so modifiziert werden, daß es schritt-minimale Beweise gibt, in denen die Terme zwischenzeitlich exponentiell größer als die Anzahl der Beweisschritte werden, siehe Letz [1993].

Es kann sogar gezeigt werden, daß es keine Datenstruktur für Terme und Klauseln *geben kann*, für die jeder PL-Resolutionsbeweis polynomiell transparent würde.

Für die AL ist aber Resolution natürlich polynomiell transparent und man kann auf dieser Basis die Kalküle miteinander vergleichen. Folgende Begriffe wurden dazu von Cook and Reckhow [1974, 1979] eingeführt.

Definition 10.1 (Nur AL) *Eine Beweisprozedur* P *ist* super *gdw sie korrekt und vollständig für KNF Formeln der AL ist, und wenn es ein Polynom* p *gibt, so daß für jede Klauselmenge* S *die Länge des kürzesten* P*-Beweises, der die Erfüllbarkeit oder Nichterfüllbarkeit von* S *anzeigt, durch* $p(|S|)$ *beschränkt ist.* ∎

Eine *deterministische* Beweisprozedur die super wäre würde natürlich $\mathsf{P} = \mathsf{NP}$ implizieren, was ihre Existenz also eher unwahrscheinlich macht.

Definition 10.2 (PL und AL) *Eine Beweisprozedur* P *kann eine Prozedur* Q p-simulieren *gdw es ein Polynom* p *gibt, so daß für alle Klauselmengen* S *gilt: wenn es einen* Q*-Beweis für* S *der Länge* n *gibt, dann gibt es einen* P*-Beweis für* S *höchstens der Länge* $p(n)$ *und wenn es keinen* Q*-Beweis für* S *gibt, dann gibt es auch keinen* P*-Beweis.*

Zwei Beweisprozeduren P *und* Q *heißen* p-äquivalent *gdw sie sich gegenseitig p-simulieren können.* ∎

Proposition 10.3 *Sei* P *nicht super und kann* Q *p-simulieren. Dann ist* Q *nicht super.*

Die Relation „kann p-simulieren" ist transitiv.

Einige der wichtigsten Ergebnisse bezüglich relativer Beweiskomplexität sind in Abbildung 10.1 (Tableaukalküle) und 10.2 (Resolutionskalküle) dargestellt.

Aussagen über relative Beweiskomplexität lassen nur in sehr beschränktem Umfang Aussagen über die Tauglichkeit von Kalkülen zu.

- Relative Beweiskomplexität ist ein indeterministisches Maß und vergleicht Kalküle, nicht (deterministische) Suchprozeduren.
- Je stärker ein Kalkül, desto größer ist im allgemeinen der Suchraum. Es gibt einen Trade-Off zwischen der Länge der kürzestmöglichen Beweise und der Größe des Suchraums. Weder sehr starke noch sehr schwache Kalküle sind nützlich wie man an den Beispielen „Tableaukalkül mit beliebigem Schnitt" und „Wahrheitstafeln" sehen kann.
- Der Suchraum von p-äquivalenten Kalkülen kann sich offenbar deutlich unterscheiden. Für die Analyse vieler Verfeinerungen ist relative Beweiskomplexität also zu grob. Ein wichtiges Beispiel sind „reguläre Konnektionstableaus mit analytischem Schnitt" und „Klauseltableaus mit analytischem Schnitt".
- Andere wichtige Eigenschaften von Kalkülen wie Zielgerichtetheit werden von der Klassifikation erst gar nicht erfaßt.

10.3 Betrachtungen über die Suchraumgröße

Bisher haben wir nur informell über die Größe des Suchraums einer Beweisprozedur gesprochen. Dies kann man formalisieren. Plaisted [1994] stellt Betrachtungen über die absolute Größe des Suchraums an. Die Idee ist, eine Beweisprozedur als einen nicht determistischen Automaten anzusehen. Für jede Eingabe

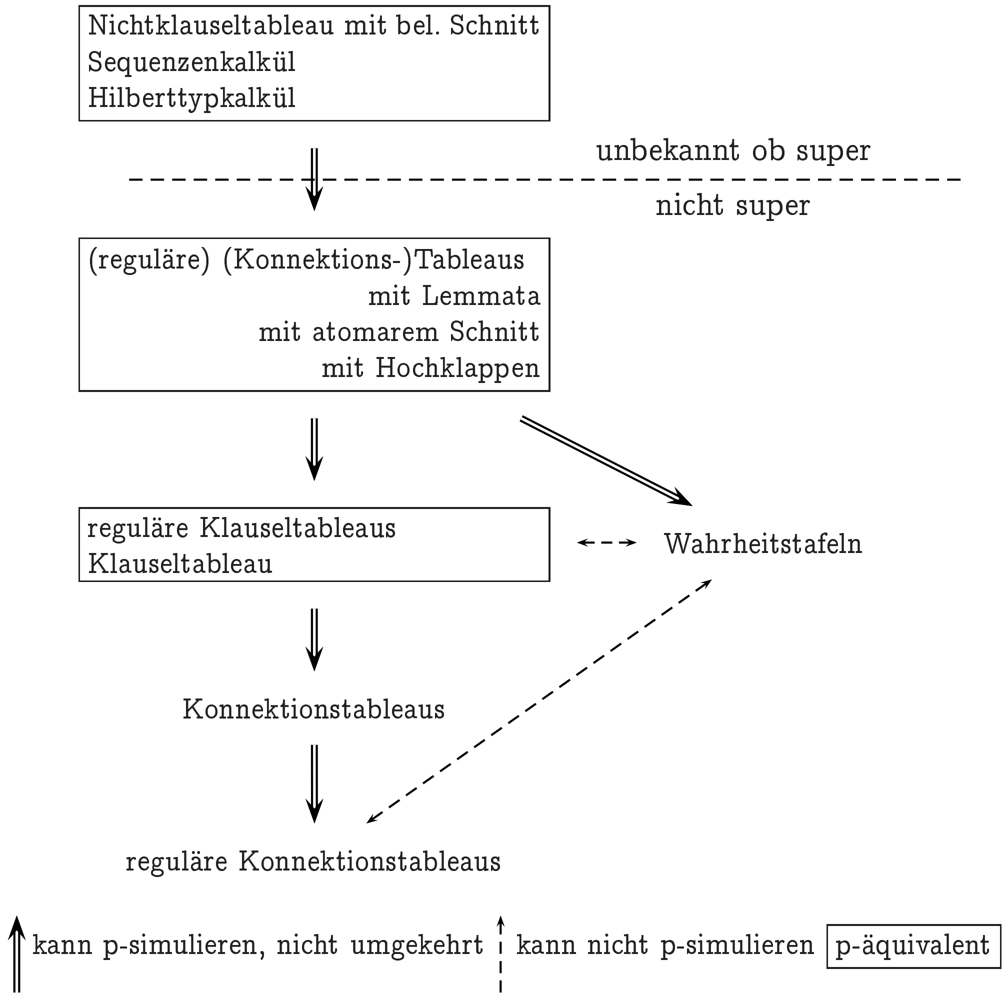


Abb. 10.1: Relative Beweiskomplexität von Tableaukalkülen.

kann man den Graph aller dafür möglichen Folgen von Zustandsübergängen betrachten.

Definition 10.4 *Eine* Beweisprozedur *ist ein Quintupel* $\langle Z, V, i, E, u \rangle$*, wobei* Z *eine Menge von Zuständen ist,* E *eine binäre Erreichbarkeitsrelation auf* Z*,* $i : For_\Sigma \rightarrow 2^Z$ *zeigt die initialen Zustände an, während* $u : S \rightarrow \{success, no\text{-}success\}$ *angibt ob ein Zustand einen Beweis darstellt. Jedes* $s \in Z$ *ist mit Elementen* $label(s) \subset V$ *markiert. Diese sind Bäume, Klauselmengen, Ketten, usw. und stellen einen Zwischenzustand der jeweiligen Beweisprozedur dar.* ■

Beispiel 10.5 Seien V zulässige Ketten, jedes $s \in Z$ mit genau einer solchen markiert und sei $(s, t) \in E$ gdw die zulässige Kette $\text{label}(t)$ durch eine Regel der

Modellelimination aus label(s) ableitbar ist. i bildet eine Menge von Klauseln S in diejenige Menge von Zuständen aus Z ab, die mit den zu $C \in S$ gehörigen Ketten markiert sind. Schließlich ist $u(s) = \text{success}$ gdw $\text{label}(s) = \langle\rangle$. ■

Mit jeder Beweisprozedur P im obigen Sinn und jeder Menge S AL-Klauseln ist der Suchraum $G(S)$ von S assoziiert wie folgt:

Definition 10.6 *Sei* $Z(S)$ *die Menge der von* $i(S)$ erreichbaren Zustände *im Graph* $\langle Z, E\rangle$, *und seien* $E(S)$ *diejenigen* $e \in E$, *die auf einem Pfad von einem* $s_0 \in i(S)$ *zu einem* $s_1 \in Z(S)$ *liegen. Der Graph* $G(S) = \langle Z(S), E(S)\rangle$ *ist der* Suchraum *von* P *für* S. ■

Definition 10.7 *In* $G(S)$ *ist die* Suchtiefe *die Länge eines maximalen Pfades, der in einem Erfolgsknoten endet.*

Die Breite *von* $G(S)$ *ist die maximale Anzahl von Zuständen aus* $Z(S)$, *die nicht auf demselben Pfad in* $G(S)$ *liegen.*

Die Knotengröße *von* $G(S)$ *ist* $\max_{s \in Z(S)} |\mathit{label}(s)|$. ■

Suchtiefe, Breite und Knotengröße heißen konstant (linear, polynomiell, exponentiell), falls sie durch eine in $|S|$ konstante (lineare, polynomielle, exponentielle) Funktion charakterisiert werden können.

Die totale *Größe des Suchraums* $G(S)$ ist $\sum_{s \in Z(S)} |\text{label}(s)|$. Sie ist durch das Produkt der Suchtiefe, Breite und Knotengröße beschränkt und kann auch als die Summe der Größe der Beweisrepräsentationen, die auf einem beliebigen von einem initialen Zustand aus erreichbaren Pfad vorkommen können, beschrieben werden.

Plaisted [1994] macht Aussagen über die maximale Größe des Suchraums für verschiedene Beweisprozeduren und AL-Hornklauseln als Eingabe.

Ein Grund dafür, daß man Hornklauseln betrachtet, ist, daß beliebige erfüllbare AL-Klauseln eine NP-vollständige Sprache sind, erfüllbare Hornklauseln dagegen können in *linearer Zeit* erkannt werden, siehe Dowling and Gallier [1984]; Gallo and Urbani [1989] (vgl. Satz 8.49 in Abschnitt 8.3.4).

Die Suchraumparameter für diesen Fall sollten also auch linear oder wenigstens polynomiell sein. Dies ist aber meist so:

Beispiel 10.8 Modellelimination hat exponentielle Suchtiefe, lineare Knotengröße und exponentielle Breite.

Die informelle Rechtfertigung für die exponentielle Suchtiefe ist, daß ME Unterziele in LIFO (last in first out) Reihenfolge bearbeitet. In einem ungünstigen Fall muß dasselbe Teilziel somit wiederholt gelöst werden. Mit Lemmata passiert dies nicht, dafür können aber die Ketten (also die Knotengröße) exponentiell werden. ■

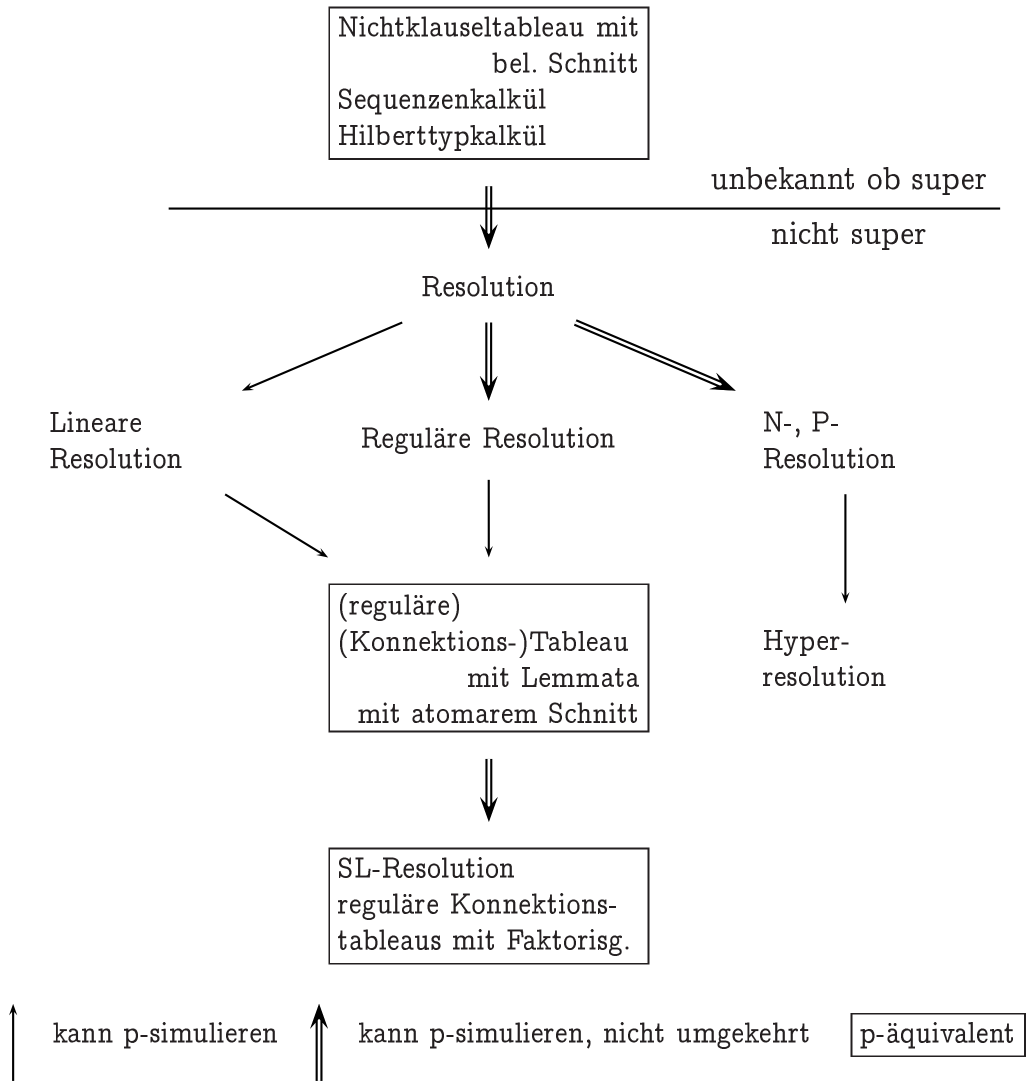


Abb. 10.2: Relative Beweiskomplexität von Resolutionskalkülen.

Konnektionstableaus (ME, SL-Resolution)	Resolution
Beweisaufzählung	Aufzählung von Klauseln (Saturierung)
Nicht beweiskonfluent	Beweiskonfluent
Gegenbeispielerzeugung	Keine Gegenbeispielerzeugung
Globale Inferenz	Lokale Inferenz
Formeln in Beweisen nicht universell	Resolventen universelle Formeln
Keine Umbenennung von Variablen	Bereinigung der Elternklauseln
Lange Beweise	Relativ kurze Beweise möglich
Kleiner lokaler Suchraum	Großer Suchraum
Sehr schwache Subsumtion	Subsumtion
Schnittfrei/analytische Schnitte	nicht analytische Schnitte (Umbenennung)
Zielorientiert	Nicht/schwach zielorientiert (SOS)
Lifting einfach	Lifting aufwendiger
Nichtklauselversionen erhältlich	Nichtklauselresolution ineffizient
Beweis erhält Formelstruktur	Beweis zerstört Formelstruktur

Tabelle 10.1: Gegenüberstellung der wichtiger Eigenschaften von konnektierten und Resolutionskalkülen.

Formale Betrachtungen über die Größe des Suchraumes haben gerade erst angefangen. Es gibt noch einige interessante offene Fragestellungen, siehe auch Bonacina [1997].

- Plaisteds Fragestellung ist noch für etliche Kalküle offen, zum Beispiel für diverse Tableauverfeinerungen.
- Man kann den Suchraum noch für andere, polynomiell entscheidbare Formelklassen betrachten, zum Beispiel 2-CNF.
- Wie kann man die Betrachtung auf volle AL oder sogar PL erweitern? Eine mögliche Antwort wäre es, in Analogie zur Beweislänge, die *relative Suchraumkomplexität* einer Beweisprozedur zu untersuchen.
- Darauf aufbauend könnte man formale Vergleiche über den Suchraum für Klausel- und Nichtklauselkalküle anstellen.

10.4 Qualitativer Vergleich von Kalkülen

Die wichtigsten Eigenschaften von konnektierten und Resolutionskalkülen sind in Tabelle 10.1 zusammengefaßt. Daraus geht hervor, daß die beiden Kalkülfamilien teilweise komplementäre Eigenschaften haben.

Beweiskonfluente Tableaukalküle erben die Eigenschaften der konnektierten Kalküle bis auf folgende Punkte:

- Erweiterung auf nicht-klassische Logiken ist einfach (bei den anderen Kalkülen ist das wegen fehlender Zielorientierung, Notwendigkeit von SNF und starker Konnektionsbedingung zum Teil problematisch).
- Sie sind nur schwach zielorientiert, dafür aber kompatibel mit Ordnungseinschränkungen.
- Gegenbeispiele können konstruiert werden.
- Kürezere Beweise, größerer (lokaler) Suchraum

Das verwendete Beweiserparadigma muß sich also nach dem Anwendungsprofil richten. Allerdings haben derzeit alle Beweisparadigmen für viele Anwendungen unakzeptable Einschränkungen. Daher zielen in letzter Zeit einige Arbeiten auf ihre *Integration* Chu and Plaisted [1993]; Ries and Hähnle [1993]; Schumann [1994].

10.5 Vergleich verschiedener Suchstrategien

In den Abbildungen 10.3 (Breitensuche), 10.4 (Tiefensuche mit Backtracking) und 10.5 (Tiefensuche mit stark vollständiger Beweisprozuedur) ist schematisch dargestellt, wie groß der Anteil des Suchraumes ist, der bei den verschiedenen Strategien durchsucht werden muß, bis ein Beweis gefunden wird.

Breitensuche ist im allgemeinen viel zu teuer (Zeit, Platz). Sie zu verwenden macht nur Sinn, wenn man den kürzesten Beweis finden will.

Bei Tiefensuche mit Backtracking wird der Suchraum bis zu einer bestimmten Tiefe durchsucht. Wenn diese Tiefe erreicht ist, ohne daß ein Beweis gefunden wurde, wird Backtracking ausgelöst. Wenn keine Choice points mehr übrig sind, wird die Tiefenschranke erhöht. Dies ist State of the art in der Implementierung von Kalkülen, die

- nicht beweiskonfluent sind,
- zwar beweiskonfluent sind aber destruktiv (dort eigentlich nicht notwendig).

Tiefensuche mit einer stark vollständigen Beweisprozedur kann nur verwendet werden, wenn der Kalkül beweiskonfluent ist. State of the art ist es, diese Strategie nur für nicht-destruktive Kalküle zu verwenden (Hypertableaus mit Range restriction, Smullyan-Tableaus, etc.). Der Informationsgehalt des (Teil-)

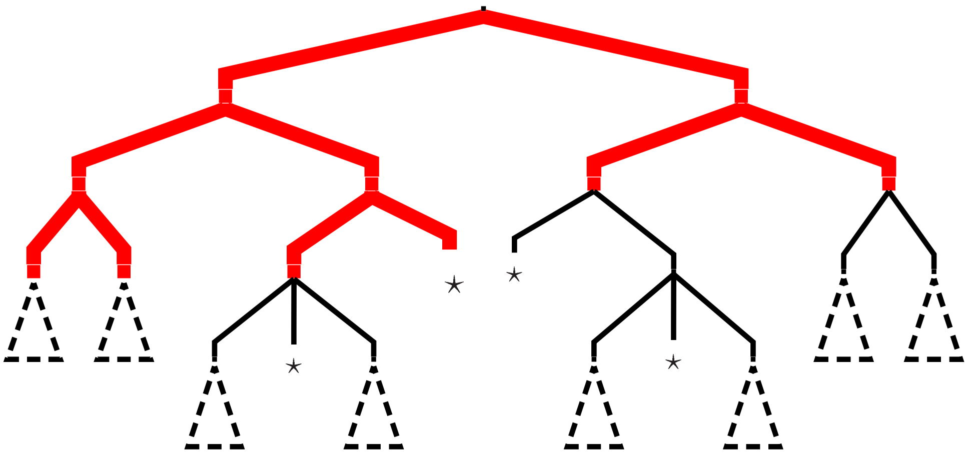


Abb. 10.3: Breitensuche

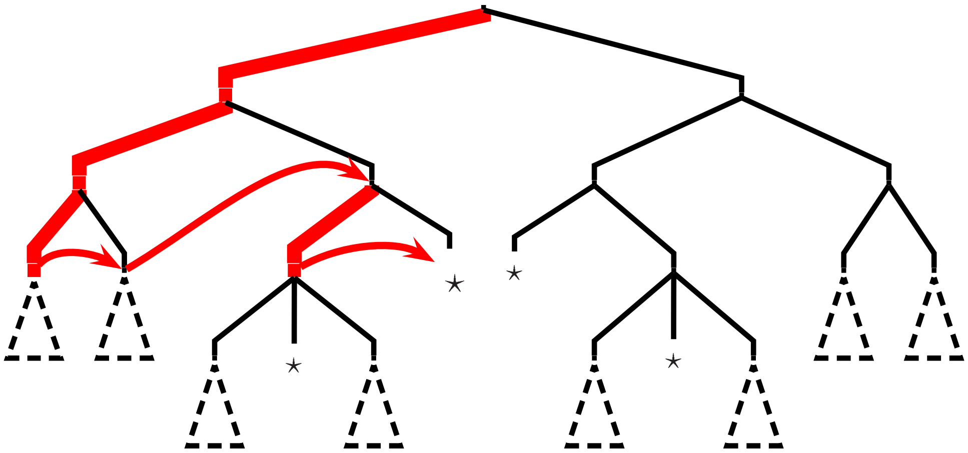


Abb. 10.4: Tiefensuche mit Backtracking.

Beweises nimmt in jedem Schritt zu; das gleiche mehrfach zu machen wird vermieden. Alle Information wird in einem Tableau gesammelt. Allerdings kann der schließlich gefundene Beweis (sehr) viel weiter unten im Suchbaum sein und entsprechend größer sein und schwerer zu finden.

Der Vergleich ist in Abbildung 10.6 noch einmal zusammengefaßt. Dort ist der Teil des Suchraums, der untersucht wird, bevor ein Beweis gefunden wird, eingefärbt. Die Form des Suchraums deutet dessen exponentielles Wachstum nach unten an.

10.6 Benchmarks

Benchmarks und Problemsammlungen sind in mancher Hinsicht problematisch.

- akademisch, realitätsfern (logische Puzzles, künstliche Probleme)

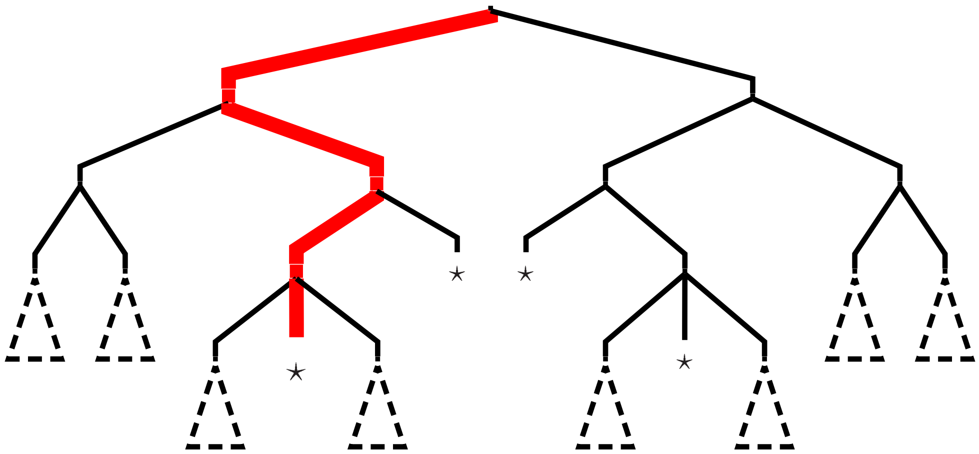


Abb. 10.5: Tiefensuche mit stark vollständiger Beweisprozudur.

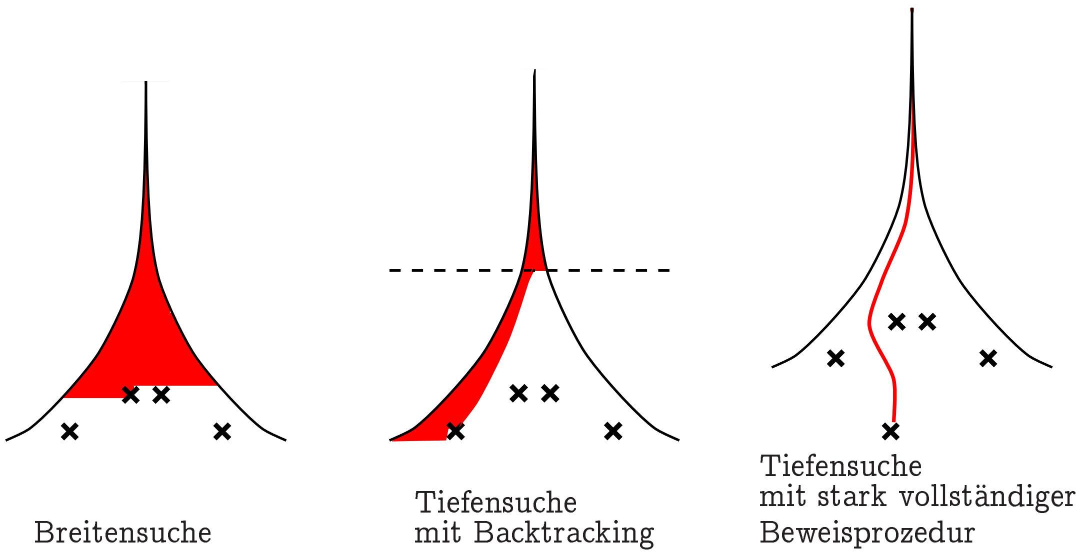


Abb. 10.6: Vergleich verschiedener Suchstrategien.

- heterogen (verschiedene Schwierigkeit und Herkunft), uninterpretiert (unklar, was formalisiert wurde), syntaktisch
- minimal unerfüllbar, irredundante Axiomatisierung
- nur allgemeingültige Formeln
- nur klassische Logik, fast nur SNF

Einige dieser Probleme wurden durch die TPTP (*Thousands of Problems for Theorem Provers*) Sammlung seit 1994 adressiert.

TPTP (2.2.1, Mai 1999) Dies ist eine Sammlung von mehr als 3000 AL- und PL-Problemen mit und ohne Gleichheit, Horn- und Nicht-Horn-Formeln in SNF, sowie ca. 200 Problemen ohne Normalform. Hervorstechende Eigenschaften sind:

- einfache, uniforme Syntax, Übersetzer in diverse andere gängige Syntaxformate werden mitgeliefert;
- verschiedene Gebiete mit Herkunftsbezeichnung und Schwierigkeit mit Einstufung;
- gut dokumentiert, regelmäßige Bug Fixes und Updates;
- unterscheidet zwischen Axiomen, Hypothesen, Theoremen

Entwickler: C. Suttner (TU München) und G. Sutcliffe (James Cook University, Australien)
Zugänglich über: `http://wwwjessen.informatik.tu-muenchen.de/~tptp/`
Dokumentation: Sutcliffe *et al.* [1994]; Suttner and Sutcliffe [1997]

10.7 Fallstudien

In Anwendungen auftretende Deduktionsprobleme sind oft spezifischer als bisher betrachtet …

- teils ist keine volle PL nötig (Hornfragment ausreichend, nur endliche Modelle möglich, …)
- viele Probleme sind einander ähnlich

… teils sind die Anforderungen aber auch allgemeiner als bisher betrachtet:

- nicht-klassische Logik ist erforderlich
- Gleichheit, eingebaute Theorien sind für die effiziente Bearbeitung notwendig

Außerdem ergeben sich noch zusätzliche Probleme und Anforderungen, die nicht von allen bzw. oder gar von keiner der bisher betrachteten Beweisprozeduren erfüllt werden:

- Viele Axiome (oft über 1000);
- hohe Redundanz der Formulierung (oft ist es notwendig, zuerst einige oft gebrauchte Folgerungen aus den Axiomen zu zeigen);
- die Formelstruktur muß erhalten bleiben (damit die Ausgabe für den Menschen verständlich bleibt);
- Simplifikation und Normalisierung von Formeln und die Gegenbeispielerzeugung werden gebraucht, und nicht nur die Widerlegung unerfüllbarer Formeln;
- das Problem und die Anwendungsdomäne enthalten nicht-logische Informationen, die verwertet werden sollten;
- Benutzerinteraktion ist für schwierige Aufgaben erforderlich;
- die Parametrisierung eines Kalküls (z.B. mit einerq Selektionsfunktion) sollte durch den Benutzer möglich sein, ohne daß dieser wirklich die Interna des Systems verstehen muß;
- „Re-use“ von Beweisen ist sinnvoll.

Eine typische Problemstellung wie sie in der Programmverifikation auftritt soll kurz dargestellt werden, um die obigen Punkte zu motivieren:

Es sei α ein Programm in einer imperativen Programmiersprache (wie z.B. Modula-2). Der Ausdruck

$$\langle\alpha\rangle\phi$$

bedeute, daß das Programm α mit der Nachbedingung ϕ terminiert[1], mithin also die totale Korrektheit von α bezüglich ϕ. Im allgemeinen hat man noch *Vorbedingungen*, so daß folgende Beweisaufgabe entsteht:

$$\psi \rightarrow \langle\alpha\rangle\phi \tag{10.1}$$

[1] Unsere Notation ist der sogenannten *dynamischen Logik* entlehnt, einer Erweiterung der Prädikatenlogik erster Stufe; siehe auch Vorlesung „Nichtklassische Logiken“.

Zum Beispiel könnte α einen Markierungsalgorithmus für gerichtete Graphen G darstellen. Dann müsste ψ allgemeine Eigenschaften und eine Axiomatisierung von gerichteten Graphen enthalten und ϕ die Aussage, daß eine Ausgabevariable von α die markierte Version von G enthält.

Man wird eine Aussage der Form (10.1) interaktiv in Teilprobleme

$$\psi' \to \langle\alpha'\rangle\phi'$$

zerlegen, so daß α' einfacher als (ein Teilprogramm von) α ist. Zum Beispiel ist $\psi \to \langle\beta;\gamma\rangle\phi$ gültig, falls $\psi \to \langle\beta\rangle\theta$ und $\theta \to \langle\gamma\rangle\phi$ gültig sind (für ein geeignetes θ).

Zwischen den Zerlegungsschritten gilt es aber immer wieder die neu entstandenen Zwischenbedingungen ϕ', ψ' zu *normalisieren* und zu *simplifizieren*.

Ist man schließlich bei rein prädikatenlogischen Verifikationsbedingungen θ' angelangt, so kann man versuchen, diese automatisch zu beweisen. Ist aber das zu verifizierende Programm α noch fehlerhaft, so sind gewisse Bedingungen gar nicht gültig. In diesem Fall kann ein *Gegenbeispiel* helfen, den Fehler in α zu finden.

Beim nächsten Verifikationsversuch mit der berichtigten Version α'' ist aber der größte Teil der Beweise *wiederverwendbar*.

Kapitel 11

Konkrete Implementierungen

In diesem Kapitel wollen wir einige konkrete Implementierungen vorstellen. Alle sind kostenlos via ftp erhältlich. Wie geben für jede der grossen Kalkülklassen (Konnektionstableaus/ME, beweiskonfluente Tableaus, Resolution) jeweils eine Vertreter an.

SETHEO 3.3 (Januar 1996)

Basiskalkül: Reguläre PL-Konnektionstableaus mit inkrementeller Tiefensuche und Backtracking.

Erweiterungen: Folding Up, Tautologielöschung, Löschung purer Klauseln; in beschränktem Maß: Subsumtion, Anti-Lemmata

Implementierung: Klauselmengen werden in den Maschinencode einer erweiterten WAM[1] (SAM) übersetzt. Übersetzer und SAM Interpretierer sind in C geschrieben. Regularität, Anti-Lemmata, Tautologielöschung durch syntaktische Constraints implementiert.

Plattformen: SunOS, Solaris, HP.

Entwickler: R. Letz, G. Stenz, A. Wolf (TU München)

Verfügbarkeit: `http://wwwjessen.informatik.tu-muenchen.de/`

Dokumentation: Letz *et al.* [1992]; Goller *et al.* [1994]

$_3$T^A^P 4.0 (Juli 1997)

Basiskalkül: PL-Nichtklauseltableaus mit inkrementeller Tiefensuche.

[1]WAM bedeutet *Warren Abstract Machine* und ist eine abstrakte Maschine mittels der die Sprache Prolog effizient implementiert wird.

Erweiterungen: Schwache Konnektionen, lokale Lemmata, Löschung purer Klauseln, universelle Formeln, eingebaute Gleichheit, mehrwertige Logik, Sorten, Pruning

Implementierung: direkt in Prolog (95 %) und C

Plattformen: SICStus Prolog 3.0.x oder Eclipse Prolog 3.5.x sowie ANSI C und Unix. Ports auf SunOS, Solaris, Linux vorhanden; andere kein Problem.

Entwickler: B. Beckert, R. Hähnle, C. Pape (Uni Karlsruhe)

Verfügbarkeit: `http://i12www.ira.uka.de/~threetap/`

Dokumentation: Beckert *et al.* [1996a,b]

OTTER 3.0.5 (Februar 1998)

Basiskalkül: PL-Resolution mit Faktorisierung

Erweiterungen: Berechnung von SNF, Löschregeln, SOS, Unit-Resolution, Hyper-Resolution, UR-Resolution, gewichtete Klauseln und Literale, universelle Theorien; in beschränktem Maß: Gleichheit, eingebaute Prädikate

Implementierung: in C über eine abstrakte Maschine (LMA: *Logic Machine Architecture*)

Plattformen: die meisten Unix Maschinen, sowie DOS und Mac

Entwickler: William McCune (Argonne National Labs, USA)

Verfügbarkeit: `http://www.mcs.anl.gov/home/mccune/ar/otter/`

Dokumentation: McCune [1994]; Wos [1996]

Waldmeister 3.0.5

Basiskalkül: Beweiser für reine Gleichheits-Logik Basiert auf Rewriting, sehr effiziente Implementierung.

Plattformen: SunOS, Solaris, Linux

Entwickler: Thomas Hillenbrand, Andreas Jaeger, Bernd Loechner, Arnim Buch (Universität Kaiserslautern)

Verfügbarkeit: `http://agent.informatik.uni-kl.de/waldmeister`

Kapitel 12

Perspektiven

12.1 Weitere Deduktionsformalismen

Klauselgraphresolution Kowalski [1975]

Man macht die Links einer Klauselmenge explizit. Sie werden nach jedem Resolutionsschritt neu berechnet und für die Steuerung des nächsten Schrittes verwendet.

Links, über die schon resolviert wurde, werden gelöscht.

Leider erwies es sich als extrem problematisch, eine terminierende und praktikable Beweisprozedur für Klauselgraphresolution zu finden, siehe Eisinger and Ohlbach [1992].

Nichtklauselresolution Murray [1982]

Seien ϕ, ψ AL-Formeln und $p \in \mathrm{Var}(\phi) \cap \mathrm{Var}(\psi)$.

Man überlegt sich leicht: wenn ϕ, ψ erfüllbar sind, dann ist es auch $\phi(p/\underline{\mathrm{true}}) \vee \psi(p/\underline{\mathrm{false}})$.

Sind ϕ, ψ Klauseln und kommt p positiv in ψ vor, dann hat man als Spezialfall dieser Beobachtung AL-Resolution.

Zusammen mit Vereinfachungsregeln (etwa: $\underline{\mathrm{true}} \wedge \underline{\mathrm{true}} \Rightarrow \underline{\mathrm{true}}$) läßt sich damit eine Nichtklauselversion von Resolution erstellen, die auf PL gehoben werden kann.

Die Steuerung einer solchen Prozedur ist recht problematisch. Auch ist die Anzahl der Resolventen enorm.

Inverse Methode Maslov [1964]

Wir gehen nicht näher darauf ein und erwähnen nur, daß sie sehr ähnlich zur Hyper-Resolution ist.

Dissolution Murray and Rosenthal [1993]

Sei ϕ eine AL-Formel in NNF in Matrixnotation (wie am Ende von Abschnitt 5.6.1 erwähnt). $\mathcal{M}$ sei ein komplementärer Pfad durch ϕ.

Die Dissolutionsregel ist eine Transformation $D : For0_\Sigma \to For0_\Sigma$, die ϕ so umbaut, daß genau der Pfad $\mathcal{M}$ aus der Matrixrepräsentation von $D(\phi)$ entfernt wird. Ist ϕ unerfüllbar, so muß nach endlich vielen Schritten $\square$ erreicht sein, da eine unerfüllbare Formel nur komplementäre Pfade enthält. Ist ϕ erfüllbar, so muß nach endlich vielen Schritten eine Formel ohne komplementäre Pfade entstanden sein.

Dissolution ist eine genuin nichtklausale Prozedur, da $D(\phi)$ nicht in KNF zu sein braucht, auch wenn ϕ es ist. Es gibt eine Version für PL.

Konsolution Eder [1991]

Eine Verallgemeinerung von Resolution und der Konnektionsmethode, vor allem für theoretische Untersuchungen bezüglich relativer Beweiskomplexität von Kalkülen in PL.

BDDs (Binary Decision Diagrams) z.B. Bryant [1986]

Sie sind eine If-then-else-Normalform für AL-Formeln, es gilt zum Beispiel:

$$\models p \wedge q \text{ gdw } \models \text{if } p \text{ then (if } q \text{ then } \underline{\text{true}} \text{ else } \underline{\text{false}}) \text{ else } \underline{\text{false}}$$

Für weitere Konnektive gibt es entsprechende Charakterisierungen. Die Herstellung einer solchen Normalform ist eng verwandt zu regulären Tableaus mit Lemmata.

In BDDs wird meist eine totale Ordnung auf Σ zugrundegelegt, außerdem werden Formeln möglichst vereinfacht und eine DAG-Repräsentation gewählt. Das Resultat nennt sich ROBDD (*Reduced Ordered BDD*).

BDDs, nicht aber ROBDDs sind auf PL hebbar Posegga [1993].

Für nichtklausale AL stellen BDDs eine der schnellsten Prozeduren zum Testen von Erfüllbarkeit dar. Sie werden in der HW-Verifikation verwendet (Vorlesung: HW-Verifikation).

Davis-Putnam-Resolution Davis and Putnam [1960]

BDDs für KNF bzw.: Fallunterscheidung und Vereinfachung durch Purity Regel und Unit-Resolution. Für AL-Formeln in KNF eine der schnellsten Prozeduren.

Hyperlinking Plaisted and Lee [1990]

Iterative Berechnung von bestimmten Instanzen einer Klauselmenge bis eine AL-Widerlegung gefunden ist. Für die Findung der Instanzen wird Unifikation verwendet.

Lineare Strategien ohne Kontrapositive Near-Horn Prolog: Loveland and Reed [1991]; Modified Problem Reduction Format (MPR, TMPR) Plaisted [1988]; Mellouli [1994]; Restart-Modellelimination: Baumgartner and Furbach [1994].

Alle sind Varianten von SLD-Resolution, die keine Kontrapositive von Klauseln verwenden; sie werden mit zusätzlichen Inferenzregeln angereichert (Restart von Goals, Fallunterscheidung), um die Vollständigkeit zu sichern.

Ihr Vorteil ist, daß sie sich effizient mit Prolog-Technologie implementieren lassen. Diese Prozeduren sind auch eng verwandt mit Klauseltableaus mit Selektionsfunktion (siehe Pape and Hähnle [1997]).

12.2 Automatisches Beweisen ohne Logik

Algebraische Methoden Die logische Äquivalenz zweier AL-Formeln ϕ und ψ kann als die Frage aufgefaßt werden, ob ϕ und ψ dasselbe Element einer Booleschen Algebra repräsentieren. Dieser Ansatz ist auf algebraisierbare nichtklassische Logiken erweiterbar.

Das Problem dabei ist, daß automatische Verfahren nur für wenige Klassen von Algebren existieren. Wenn es aber welche gibt, können sie oft mehr hergeben als man mit logischen Verfahren erhält, zum Beispiel, Normalisierung oder Gegenbeispiele.

Lineare Optimierung z.B.: Hooker [1988]

Eine Klausel der Form

$$\{p_1, \ldots, p_i, \neg p_{i+1}, \ldots, \neg p_n\}$$

ist offenbar erfüllbar gdw die lineare Ungleichung

$$p_1 + \cdots + p_i + (1 - p_{i+1}) + \cdots + (1 - p_n) \geq 1$$

über $\{0, 1\}$ lösbar ist.

Das ergibt eine Reduktion der Erfüllbarkeit AL-Klauseln auf die Lösbarkeit von linearen, ganzzahligen Optimierungsproblemen (*Integer programming*).

Falls die Signatur des Problems durch $\Sigma = \{p_1, \ldots, p_n\}$ gegeben ist, werden durch die Kostenfunktion $\Sigma_{i=1}^{n} p_i$, die minimiert wird, minimale Modelle berechnet.

Evolutionäre Algorithmen Gent and Walsh [1993]; Selman *et al.* [1992]

Eine Gütefunktion f ist für Interpretationen von AL-Formeln Φ in KNF definiert, zum Beispiel

$$f(I, \Phi) = \frac{1}{2^{u_I}} \ ,$$

wobei u_I die Anzahl der von I nicht erfüllten Klauseln in Φ ist.

Falls Φ erfüllbar ist, finden evolutionäre Algorithmen meist schnell eine erfüllende Interpretation. Sie können allerdings nicht zum Nachweis der Unerfüllbarkeit eingesetzt werden.

12.3 Automatisches Beweisen in Nichtklassischen Logiken und Erweiterungen der PL

Sortierte Logik „typisierte Prädikatenlogik“

Objektvariablen, Prädikate und Funktionen müssen deklariert werden, zum Beispiel:

```
p: zahl;
(∀x:zahl) p(x)
...
```

Jeder Term hat eine *Sorte*. Die Sorten können untereinander über eine Untersortenrelation geordnet sein.

Damit können sinnlose Instantiierungen (etwa Strings, wo Zahlen stehen müssten) vermieden werden. Dies kann zu einer erheblichen Einschränkung des Suchraums führen.

Technisch wird die Sorteneinschränkung dadurch implementiert, daß bei der Unifikation nur mit den Sorten der Terme kompatible Substitutionen zugelassen werden. Für bestimmte Sortenhierarchien kann dies zur Nichteindeutigkeit von mgus führen.

Einige Artikel zu sortierter Logik und Literaturhinweise finden sich in dem Band Bläsius *et al.* [1989], speziell zu Tableaus siehe Weidenbach [1995].

Logik höherer Stufe[1]: Die Quantifizierung über Funktions- und Prädikatensymbole wird erlaubt.

Im Gegensatz zu PL erster Stufe lassen sich zum Beispiel die natürlichen Zahlen in Logik zweiter Stufe bis auf isomorphe Modelle eindeutig charakterisieren, indem man folgendes Induktionsaxiom hinzunimmt:

$$(\forall p)((p(0) \land (\forall x)p(x) \to p(s(x))) \to (\forall x)p(x))$$

Die Literatur zu HOL ist riesig. Eine für das Automtische Beweisen relevante Einführung ist von Leivant [1994]. (Vorlesung: HW-Verifikation)

Mehrwertige Logik Es gibt weitere Wahrheitswerte, außer den zwei üblichen, z.B.: $\{F, \underline{\text{undefined}}, W\}$.

Zum Beispiel kann die Semantik rekursiver Prädikate, die ja nur partiell definiert zu sein brauchen, mit einer solchen Logik elegant charakterisiert werden, Kerber and Kohlhase [1996]. Dann muß

$$(\forall x)p(x) \lor \neg p(x)$$

keine allgemeingültige Formel sein. Der Wahrheitswert „undefined“ für ein Grundatom p hat dann die Bedeutung „p terminiert nicht/ist undefiniert für Eingabe t“.

Literatur: Hähnle [1994] (Vorlesung: Nichtklassische Logiken)

Modal- und Temporallogik $\phi \to \Diamond\psi$ bedeutet: wenn jetzt ϕ gilt, dann gilt irgendwann ψ.

(Vorlesung: Nichtklassische Logiken)

Literatur: Hughes and Cresswell [1984]; Wolper [1985]; Burch *et al.* [1990]; Stirling [1992]

Nicht-monotone Logik $\Phi \models \psi$, aber nicht unbedingt $\Phi \cup \Theta \models \psi$.

(Vorlesung: Nichtklassische Logiken; Literatur ist im Skriptum zu dieser Vorlesung angegeben)

12.4 Taktisches Theorembeweisen

Eine *Taktik* ist eine Makroinferenzregel zusammen mit einer Vorschrift, wie ein Beweis dafür (mit einfacheren Inferenzregeln) zu finden ist. An einigen Stellen ist evtl. Benutzerinteraktion erforderlich.

[1] Engl.: higher order logic (HOL).

Komplexe Beweisaufgaben (wie formale Korrektheitsnachweise von Programmen) können nicht durch Suche vollautomatisch gelöst werden. Stattdessen werden „Beweisprogrammiersprachen“ benutzt. Prozeduren in solchen Sprachen werden *Taktiken* genannt.

Der Entwurf angemessener Taktiken für ein gegebenes Problem ist eine anspruchsvolle Aufgabe. Ein aktuelles Forschungsziel ist die Integration von automatischem und interaktivem Beweisen.

Wie schon erwähnt ergeben sich zusätzliche Aufgaben für automatische Beweiser: Erzeugung von Gegenbeispielen, Vereinfachung, Normalisierung.

(Praktikum: Programm-Verifikation)

12.5 Spezialbeweiser

Geometrie

Fragmente der Arithmetik Zum Beispiel *Presburger Arithmetik:* PL mit der Signatur

$$\langle\{0/0, s/1\}, \{plus/2, \doteq/2\}\rangle$$

über der Theorie der natürlichen Zahlen ist entscheidbar. Dafür gibt es spezielle Verfahren.

Literaturverzeichnis

R. Anderson and W. Bledsoe. A linear format for resolution with merging and a new technique for establishing completeness. *Journal of the ACM*, 17(3):525–534, July 1970. Reprinted in Siekmann and Wrightson [1983c].

Peter B. Andrews. Theorem proving through general matings. *Journal of the ACM*, 28:193–214, 1981.

Irving Anellis. From semantic tableaux to Smullyan trees: the history of the falsifiability. *Modern Logic*, 1(1):36–69, June 1990.

Irving Anellis. Erratum, from semantic tableaux to Smullyan trees: the history of the falsifiability. *Modern Logic*, 2(2):219, December 1991.

Kristoff R. Apt and Maarten H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3):841–862, 1982.

Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

Leo Bachmair and Harald Ganzinger. A theory of resolution. Technical Report MPI–I–97–2–005, Max-Planck-Institut für Informatik, Saarbrücken, April 1997. To appear in: J. A. Robinson & A. Voronkov (eds.), *Handbook of Automated Reasoning*, Elsevier.

Peter Baumgartner and Uli Furbach. Model Elimination without Contrapositives and its Application to PTTP. *Journal of Automated Reasoning*, 13:339–359, 1994.

Bernhard Beckert and Reiner Hähnle. Analytic tableaux. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction — A Basis for Applications*, volume I: Foundations, pages 11–41. Kluwer, Dordrecht, 1998.

Bernhard Beckert and Joachim Posegga. leanTAP: Lean tableau-based deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.

Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt. The *even more* liberalized δ-rule in free variable semantic tableaux. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Proceedings of the third Kurt Gödel Colloquium KGC'93, Brno, Czech Republic*, volume 713 of *LNCS*, pages 108–119. Springer-Verlag, August 1993.

Bernhard Beckert, Reiner Hähnle, Karla Geiß, Peter Oel, Christian Pape, and Martin Sulzmann. The Many-Valued Tableau-Based Theorem Prover ${}_3T^AP$, Version 4.0. Interner Bericht 3/96, Universität Karlsruhe, Fakultät für Informatik, 1996.

Bernhard Beckert, Reiner Hähnle, Peter Oel, and Martin Sulzmann. The tableau-based theorem prover ${}_3T^AP$, version 4.0. In Michael McRobbie and John Slaney, editors, *Proc. 13th Conference on Automated Deduction, New Brunswick/NJ, USA*, volume 1104 of *LNCS*, pages 303–307. Springer-Verlag, 1996.

Wolfgang Bibel and Elmar Eder. Methods and calculi for deduction. In Dov M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1: Logical Foundations, pages 67–182. Oxford University Press, 1992.

Wolfgang Bibel. Tautology testing with a generalized matrix method. *Theoretical Computer Science*, 8:31–44, 1979.

Wolfgang Bibel. On matrices with connections. *Journal of the ACM*, 28:633–645, 1981.

Wolfgang Bibel. *Automated Theorem Proving*. Vieweg, Braunschweig, second revised edition, 1987.

K..-H. Bläsius, U. Hedtstück, and C. Rollinger, editors. *Sorts and Types in Artificial Intelligence*, volume 418 of *LNAI*. Springer-Verlag, April 1989.

Maria Paola Bonacina. A note on the analysis of theorem-proving strategies. *Association for Automated Reasoning (AAR) Newsletter*, 36:2–8, April 1997.

Robert S. Boyer. *Locking: A Restriction of Resolution*. PhD thesis, The University of Texas at Austin, 1971.

Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Sequential circuit verification using symbolic model checking. In *Proc. 27th Design Automation Conference (DAC 90)*, pages 46–51, 1990.

Ricardo Caferra and Nicolas Zabel. A tableaux method for systematic simultaneous search for refutations and models using equational problems. *Journal of Logic and Computation*, 3(1):3–26, 1993.

Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, London, 1973.

Chin-Liang Chang. The unit proof and the input proof in theorem proving. *Journal of the ACM*, 17(4):698–707, 1970.

Heng Chu and David A. Plaisted. Rough resolution: A refinement of resolution to remove large literals. In *Proc. 11th National Conference on Artificial Intelligence, Washington/DC, USA*, pages 15–20, 1993.

Stephen Cook and Robert Reckhow. On the lengths of proofs in the propositional calculus. In *Proceedings 6th STOC*, pages 135–148, 1974.

Stephen Cook and Robert Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36ff, 1979.

Marcello D'Agostino. Are tableaux an improvement on truth tables? Cut-free proofs and bivalence. *Journal of Logic, Language and Information*, 1:235–252, 1992.

Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

Martin Davis. The prehistory and early history of automated deduction. In Jörg Siekmann and Graham Wrightson, editors, *Automation of Reasoning: Classical Papers in Computational Logic 1957–1966*, volume 1, pages 1–28. Springer-Verlag, 1983.

W. Dowling and J. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulæ. *Journal of Logic Programming*, 3:267–284, 1984.

Elmar Eder. Consolution and its relation with resolution. In *Proc. IJCAI, International Joint Conference on Artificial Intelligence*, 1991.

Norbert Eisinger and Hans Jürgen Ohlbach. Deduction systems based on resolution. In Dov M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1: Logical Foundations, pages 182–271. Oxford University Press, 1992.

C. Fermüller, A. Leitsch, T. Tammet, and N. Zamov. *Resolution Methods for the Decision Problem.* LNAI 679. Springer-Verlag, 1993.

Melvin C. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, New York, 1990.

Melvin C. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, New York, second edition, 1996.

Hiroshi Fujita and Ryuzo Hasegawa. A model generation theorem prover in KL1 using a ramified-stack algorithm. In Koichi Furukawa, editor, *Proceedings 8th International Conference on Logic Programming, Paris/France*, pages 535–548. MIT Press, 1991.

Jean H. Gallier. *Logic for Computer Science: Foundations of Automated Theorem Proving.* Harper and Row, New York, 1986.

Giorgio Gallo and Giampaolo Urbani. Algorithms for testing the satisfiability of propositional formulae. *Journal of Logic Programmming*, 7(1):45–62, March 1989.

M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, San Francisco, 1979.

Ian P. Gent and Toby Walsh. Towards an understanding of hill-climbing procedures for SAT. Technical report, Dept. of AI, University of Edinburgh, 1993.

Andreas Goerdt. Regular resolution versus unrestricted resolution. *SIAM Journal of Computing*, 22(4), 1993.

Christian Goller, Reinhold Letz, Klaus Mayr, and Johann Schumann. SETHEO V3.2: recent developments. In Alan Bundy, editor, *Proc. 12th Conference on Automated Deduction CADE, Nancy/France*, LNAI 814, pages 778–782. Springer-Verlag, 1994.

G. Gottlob and A. Leitsch. On the efficiency of subsumtion algorithms. *Journal of the ACM*, 32(2):280–295, 1985.

Reiner Hähnle and Stefan Klingenbeck. A-ordered tableaux. *Journal of Logic and Computation*, 6(6):819–834, 1996.

Reiner Hähnle and Christian Pape. Ordered tableaux: Extensions and applications. In Didier Galmiche, editor, *Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Pont-à-Mousson, France*, volume 1227 of *LNCS*, pages 173–187. Springer-Verlag, 1997.

Reiner Hähnle and Peter H. Schmitt. The liberalized δ-rule in free variable semantic tableaux. *Journal of Automated Reasoning*, 13(2):211–222, October 1994.

Reiner Hähnle. *Automated Deduction in Multiple-Valued Logics*, volume 10 of *International Series of Monographs on Computer Science*. Oxford University Press, 1994.

Robert Hill. LUSH-resolution and its completeness. DCL Memo 78, Department of Artificial Intelligence, University of Edinburgh, August 1974.

John N. Hooker. A quantitative approach to logical inference. *Decision Support Systems*, 4:45–69, 1988.

G. E. Hughes and M. J. Cresswell. *A Companion to Modal Logic*. Methuen Press, 1984.

William H. Joyner. Resolution strategies as decision procedures. *Journal of the ACM*, 23(3):398–417, 1976.

Stig Kanger. *Provability in Logic*, volume 1 of *Acta Universitatis Stockholmiensis*. Almqvist & Wiksell, Stockholm, 1957.

Manfred Kerber and Michael Kohlhase. A tableau calculus for partial functions. In *Collegium Logicum. Annals of the Kurt-Gödel-Society*, volume 2, pages 21–49. Springer-Verlag, Wien New York, 1996.

Hans Kleine Büning and Theodor Lettmann. *Aussagenlogik: Deduktion und Algorithmen*. B. G. Teubner, Stuttgart, 1994.

Stefan Klingenbeck and Reiner Hähnle. Semantic tableaux with ordering restrictions. In Alan Bundy, editor, *Proc. 12th Conference on Automated Deduction CADE, Nancy/France*, volume 814 of *LNCS*, pages 708–722. Springer-Verlag, 1994.

R. Kowalski and P. J. Hayes. Semantic trees in automatic theorem proving. In *Machine Intelligence*, volume 4, pages 87–101. Edinburgh University Press, 1969. Reprinted in Siekmann and Wrightson [1983c].

Robert Kowalski and Donald Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2(3):227–260, 1971.

Robert Kowalski. Predicate logic as a programming language. *Information Processing*, 74:569–574, 1974.

Robert Kowalski. A proof procedure using connection graphs. *Journal of the ACM*, 22(4):572–595, 1975.

Alexander Leitsch. *The Resolution Calculus*. EATCS Texts in Theoretical Computer Science. Springer-Verlag, 1997.

Daniel Leivant. Higher order logic. In Dov M. Gabbay, C.J̃. Hogger, and J.Ã. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1: Deduction Methodologies, pages 229–322. Oxford University Press, 1994.

Reinhold Letz, Johann Schumann, Stephan Bayerl, and Wolfgang Bibel. SETHEO: A high-perfomance theorem prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.

Reinhold Letz. *First-Order Calculi and Proof Procedures for Automated Deduction*. PhD thesis, TH Darmstadt, June 1993.

John W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, Second edition, 1987.

Donald W. Loveland and David W. Reed. A near-Horn Prolog for compilation. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT Press, Cambridge, MA, 1991.

Donald W. Loveland. Mechanical theorem proving by model elimination. *Journal of the ACM*, 15(2):236–251, April 1968. Reprinted in: Siekmann and Wrightson [1983a].

Donald W. Loveland. A simplified format for the model elimination procedure. *Journal of the ACM*, 16(3):349–363, July 1969. Reprinted in: Siekmann and Wrightson [1983c].

Donald W. Loveland. A unifying view of some linear Herbrand procedures. *Journal of the ACM*, 19(2):366–384, 1972.

Donald W. Loveland. *Automated Theorem Proving. A Logical Basis*, volume 6 of *Fundamental Studies in Computer Science*. North-Holland, 1978.

Donald W. Loveland. Automated theorem proving: A quarter century review. *Contemporary Mathematics*, 29:1–45, 1984.

D. Luckham. Refinements in resolution theory. In *Proc. IRIA Symp. Automatic Demonstration*, pages 163–190, Versailles, France, 1968. Springer-Verlag. Reprinted in Siekmann and Wrightson [1983c].

Donald Mackenzie. The automation of proof: a historical and sociological exploration. *IEEE Annals of the History of Computing*, 17(3):7–29, 1995.

Rainer Manthey and François Bry. SATCHMO: A theorem prover implemented in Prolog. In Ewing Lusk and Ross Overbeek, editors, *Proceedings 9th Conference on Automated Deduction*, LNCS, New York, pages 415–434. Springer-Verlag, 1988.

A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.

S. Ju. Maslov. An inverse method of establishing deducibilities in the classical predicate calculus. *Dokl. Akad. Nauk SSSR*, 159:1420–1424, 1964. Reprinted in Siekmann and Wrightson [1983b].

S. Ju. Maslov. Proof-search strategies for methods of resolution type. In *Machine Intelligence*, volume 6, pages 77–90. Elsevier, 1971.

J. D. McCharen, R. A. Overbeek, and L. Wos. Problems and experiments for and with automated theorem-proving programs. *IEEE Trans. on Computers*, C-25(8):773–782, 1976.

William W. McCune. *OTTER 3.0 Reference Manual and Guide*. Argonne National Laboratory/IL, USA, 1994.

Taïeb Mellouli. TMPR3: A tree representation of the modified problem reduction format and its extension to three-valued logic. *Journal of Automated Reasoning*, 1994.

Neil V. Murray and Erik Rosenthal. Dissolution: Making paths vanish. *Journal of the ACM*, 40(3):504–535, 1993.

Neil V. Murray. Completely non-clausal theorem proving. *Artificial Intelligence*, 18:67–85, 1982.

Richard A. O'Keefe. *The Craft of Prolog*. The MIT Press, Cambridge, 1990.

Christian Pape and Reiner Hähnle. Restart tableaux with selection function. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Fifth Kurt-Gödel-Colloquium, KGC'97, Vienna*, volume 1289 of *LNCS*, pages 219–232. Springer-Verlag, 1997.

David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

David A. Plaisted and Shie-Jue Lee. Inference by clause matching. In Zbigniew Raś and Maria Zemankova, editors, *Intelligent Systems—State of the art and future directions*, chapter 8, pages 200–235. Ellis Horwood, 1990.

David A. Plaisted. Non-Horn clause logic programming without contrapositives. *Journal of Automated Reasoning*, 4:287–325, 1988.

David Plaisted. The search efficiency of theorem proving strategies. In Alan Bundy, editor, *Proc. 12th Conference on Automated Deduction CADE, Nancy/France*, LNAI. Springer-Verlag, 1994.

Joachim Posegga. *Deduktion mit Shannongraphen für Prädikatenlogik erster Stufe*. PhD thesis, University of Karlsruhe, 1993. Diski 51, infix Verlag.

Dag Prawitz. An improved proof procedure. *Theoria*, 26:102–139, 1960. Reprinted in Siekmann and Wrightson [1983b].

Steve Reeves. Semantic tableaux as a framework for automated theorem-proving. In C. S. Mellish and J. Hallam, editors, *Advances in Artificial Intelligence (Proceedings of AISB-87)*, pages 125–139. Wiley, 1987.

Ray Reiter. Two results on ordering for resolution with merging and linear format. *Journal of the ACM*, 18:630–646, March 1971.

Klaus Ries and Reiner Hähnle. Prädikatenlogisches Beweisen mit gemischt ganzzahliger Optimierung. Ein tableaubasierter Ansatz. In *Working Notes of Workshop Künstliche Intelligenz und Operations Research, Berlin (published as Tech Report, Max-Planck-Institut für Informatik, Saarbrücken, MPI-I-93-234)*, September 1993.

J. A. Robinson. Automatic deduction with hyper-resolution. *Int. Journal of Computer Math.*, 1:227–234, 1965. Reprinted in Siekmann and Wrightson [1983b].

J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965. Reprinted in Siekmann and Wrightson [1983b].

J. A. Robinson. A review of automatic theorem-proving. In *Proc. Annual Symposium in Applied Mathematics XIX*, pages 1–18, American Mathematical Society, Providence, 1967.

J. A. Robinson. The generalized resolution principle. In *Machine Intelligence*, volume 3, pages 77–93. Oliver and Boyd, Edinburgh, 1968. Reprinted in Siekmann and Wrightson [1983c].

J. A. Robinson. Logic and logic programming. *CACM*, 35(3):40–65, March 1992.

Peter H. Schmitt. The THOT theorem prover. Technical Report TR–87.09.007, IBM Heidelberg Scientific Center, 1987.

Peter H. Schmitt. Formale Systeme. Vorlesungsskriptum Fakultät für Informatik, Universität Karlsruhe, Wintersemester 1992.

Peter H. Schmitt. *Theorie der logischen Programmierung*. Springer-Verlag, 1992.

Uwe Schöning. *Logik für Informatiker*. Number 56 in Informatik. Bibliographisches Institut Mannheim, 3., überarb. edition, 1992.

Johann Ph. Schumann. DELTA — a bottom-up processor for top-down theorem provers (system abstract). In Alan Bundy, editor, *Proc. 12th Conference on Automated Deduction CADE, Nancy/France*, LNAI. Springer-Verlag, 1994.

Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proc. of AAAI-92, San Jose/CA, USA*, pages 440–446, 1992.

Jörg Siekmann and Graham Wrightson, editors. *Automation of Reasoning: Classical Papers in Computational Logic*. Springer, 1983.

Jörg Siekmann and Graham Wrightson, editors. *Automation of Reasoning: Classical Papers in Computational Logic 1957–1966*, volume 1. Springer-Verlag, 1983.

Jörg Siekmann and Graham Wrightson, editors. *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 2. Springer-Verlag, 1983.

James R. Slagle. Automatic theorem proving with renamable and semantic resolution. *Journal of the ACM*, 14(4):687–697, 1967. Reprinted in Siekmann and Wrightson [1983c].

Raymond M. Smullyan. *First-Order Logic*. Dover Publications, New York, second corrected edition, 1995. First published 1968 by Springer-Verlag.

Leon Sterling and Ehud Shapiro. *The Art of Prolog*. MIT Press, Cambridge, Massachusetts, 1986.

Mark E. Stickel. A Prolog technology theorem prover: A new exposition and implementation in Prolog. *Theoretical Computer Science*, 104(1):109–129, 1992.

Colin Stirling. Modal and temporal logics. In S. Abramsky, Dov M. Gabbay, and T. S.Ê. Maibaum, editors, *Handbook of Logic in Computer Science. Volume 2. Background: Computational Structures*, pages 477–563. Oxford University Press, 1992.

Geoff Sutcliffe, Christian Suttner, and Theodor Yemenis. The TPTP problem library. In Alan Bundy, editor, *Proc. 12th Conference on Automated Deduction CADE, Nancy/France*, LNAI, pages 252–266. Springer-Verlag, 1994.

Christian Suttner and Geoff Sutcliffe. The TPTP problem library: TPTP v2.0.0. Technical Report AR-97-01, Institut für Informatik, TU München, 1997.

G. Tseitin. On the complexity of proofs in propositional logics. *Seminars in Mathematics*, 8, 1970. Reprinted in Siekmann and Wrightson [1983c].

Tie-Cheng Wang and W. W. Bledsoe. Hierarchical deduction. *Journal of Automated Reasoning*, 3(1):35–77, 1987.

Christoph Weidenbach. First-order tableaux with sorts. *Journal of the Interest Group in Pure and Applied Logics*, 3(6):886–907, 1995.

Pierre Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 28(110–111):119–136, June–September 1985.

L. Wos, D. Carson, and G. A. Robinson. The unit preference strategy in theorem proving. In *Fall Joint Computer Conference, AFIPS, Washington D.C.*, pages 615–621. Spartan Books, 1964. Reprinted in Siekmann and Wrightson [1983b].

L. Wos, G. A. Robinson, and D. F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the ACM*, 12(4):536–541, 1965. Reprinted in Siekmann and Wrightson [1983b].

Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle. *Automated Reasoning—Introduction and Applications.* Prentice-Hall, Englewood Cliffs, 1984.

Larry Wos. *The Automation of Reasoning: An Experimenter's Notebook with OTTER Tutorial.* Academic Press, 1996.

Index