Kapitel II:
Grundlagen und Beispiele

(Norbert Eisinger & Hans Jürgen Ohlbach)

1 Einführung

Ein Deduktionssystem ist ein programmierbares Verfahren zur Erkennung von Folgerungsbeziehungen zwischen Aussagen. Die Folgerungsbeziehung sei durch ein typisches Beispiel illustriert, dessen generelle Form bereits auf Aristoteles zurückgeht:

Aussage 1:	„Alle Menschen sind sterblich“
Aussage 2:	„Sokrates ist ein Mensch“
_____	_____
Aussage 3:	„Sokrates ist sterblich“.

In diesem Fall folgt Aussage 3 aus den ersten beiden Aussagen. Sobald man die ersten beiden als wahr annimmt, muß man zwangsläufig auch die dritte als wahr akzeptieren.

Es gibt verschiedene Varianten, die genaue Aufgabe eines Deduktionssystems zu spezifizieren. Sie kann zum Beispiel darin bestehen, zu entscheiden, ob eine gegebene Aussage aus anderen gegebenen folgt. Oder das Deduktionssystem soll Folgerungsbeziehungen zwischen gegebenen Aussagen nachweisen, wenn diese Beziehungen tatsächlich bestehen. Eine weitere Möglichkeit wäre, daß das Deduktionssystem von gegebenen Aussagen ausgehen und daraus neue Aussagen erzeugen soll, die aus den gegebenen folgen.

Die Mechanismen, die einen Rechner zur Lösung solcher Aufgaben befähigen, sind im Prinzip sehr ähnlich zu denen, die ihn zum Rechnen mit Zahlen befähigen. Die Aussagen müssen als Datenobjekte repräsentiert und die Schlußfolgerungen als Operationen auf diesen Datenobjekten programmiert werden.

Ein komplettes Deduktionssystem baut sich im wesentlichen aus vier Schichten auf.

Die unterste Schicht wird durch eine *Logik* gebildet, die mit der Festlegung der Syntax einer formalen Sprache und deren Semantik die zulässige Struktur und die Bedeutung von Aussagen vorgibt. Aussagen entsprechen Formeln der Logik. Die gewählte Logik bestimmt ganz konkret, welche Arten von Aussagen erlaubt und welche verboten sind. Beispielsweise kann sie festlegen, daß eine Quantifizierung „Für alle Zahlen gilt …“ erlaubt, eine Quantifizierung „Für alle stetigen Funktionen gilt …“ dagegen verboten sein soll. Die Definition einer Bedeutung für die Formeln liefert in den Logiken, die für Deduktionssysteme interessant sind, darüberhinaus eine Beziehung „aus $\mathcal{A}$ folgt $\mathcal{B}$“ zwischen Aussagen $\mathcal{A}$ und $\mathcal{B}$. Damit ist ein semantischer *Folgerungsbegriff* etabliert und formal präzise definiert. Diese Definition hilft jedoch zunächst in keiner Weise, für gegebene $\mathcal{A}$ und $\mathcal{B}$ algorithmisch zu bestimmen, ob $\mathcal{B}$ wirklich aus $\mathcal{A}$ folgt.

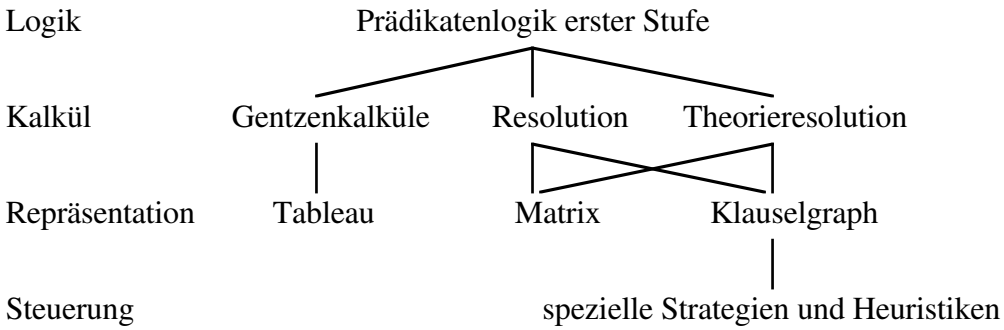
Dies ist Gegenstand des *Kalküls*, der zweiten Schicht eines Deduktionssystems. Ein Kalkül definiert syntaktische *Ableitungen* als Operationen auf den Formeln. Damit kann aus einer

Formel $\mathcal{A}$ durch reine Symbolmanipulation eine Formel $\mathcal{B}$ gewonnen werden, wobei die Bedeutung der in $\mathcal{A}$ und $\mathcal{B}$ vorkommenden Symbole überhaupt keine Rolle spielt. Im obigen Beispiel sollte man also nicht wissen müssen, wer oder was Sokrates eigentlich ist, um die Ableitung durchzuführen. Ein syntaktisch aus $\mathcal{A}$ abgeleitetes $\mathcal{B}$ soll aber trotzdem semantisch folgen und umgekehrt. Ein korrekter Kalkül stellt daher nur solche Ableitungsoperationen zur Verfügung, die garantieren, daß alles syntaktisch Ableitbare auch semantisch folgt. Wenn umgekehrt alles, was semantisch folgt, auch syntaktisch ableitbar ist, ist der Kalkül vollständig. Nach dem berühmten Unvollständigkeitssatz von Kurt Gödel sind vollständige Kalküle ab einer gewissen Ausdrucksstärke der Logiken jedoch nicht möglich.

Die dritte Schicht eines Deduktionssystems, die *Repräsentation*, bestimmt die Darstellung von Formeln oder Formelmengen, von deren Beziehungen zueinander, sowie von den jeweiligen Zuständen der Ableitungsketten. Auf dieser Ebene werden oft zusätzliche Operationen eingeführt, etwa das Löschen von redundanten Aussagen, so daß im Kalkül noch mögliche Ableitungen nun nicht mehr möglich – und hoffentlich auch nicht mehr nötig – sind. Eine gute Repräsentationsschicht trägt ganz entscheidend zur Effizienz eines Deduktionssystems bei.

Die letzte Schicht eines Deduktionssystems, die *Steuerung*, enthält schließlich die Strategien und Heuristiken, mit denen unter den möglichen Ableitungsschritten die jeweils sinnvollen ausgewählt werden. Hier steckt die eigentliche „Intelligenz“ des Systems.

Es ist in diesem Rahmen nicht möglich, alle bisher entwickelten Komponenten für Deduktionssysteme vorzustellen. Wir beschränken uns auf einige der am besten untersuchten und am häufigsten angewandten Methoden, welche gleichzeitig Modellcharakter für andere Systeme haben:



Die Theorieresolution stellt ein allgemeines Schema dar, um Kalküle für spezielle Anwendungen zu konstruieren. Fast alle in den übrigen Beiträgen vorgestellten Verfahren kann man als Theorieresolution für besonders wichtige Anwendungsfälle auffassen.

2 Prädikatenlogik erster Stufe

Die Prädikatenlogik erster Stufe (PL1) ist eine Erweiterung der im letzten Jahrhundert von George Boole entwickelten und heute als die logische Basis der Digitaltechnik geltenden Aussagenlogik (PL0). Eine graphisch orientierte Syntax für PL1 wurde ebenfalls bereits im letzten Jahrhundert von Gottlob Frege in seiner „Begriffsschrift“ vorgestellt [Fre79]. Die heute

übliche Syntax geht auf Giuseppe Peano zurück. Eine Semantik und zugehörige Kalküle wurden in der ersten Hälfte dieses Jahrhunderts von Alfred Tarski, David Hilbert, Gerhard Gentzen und anderen Logikern entwickelt. Die Prädikatenlogik erster Stufe ist heute die am besten untersuchte und in der Anwendung am weitesten verbreitete Logik. Sie dient darüberhinaus als Basis für viele andere Logiken – Modallogiken, Temporallogiken, dynamische Logiken etc. –, die in der Künstlichen Intelligenz und bei der Beschreibung von Programmen und Prozessen eine Rolle spielen. Die folgenden Abschnitte fassen Syntax und Semantik von PL1 informell zusammen und führen die gebräuchliche Terminologie ein. Genauere Definitionen kann man in jedem Logiklehrbuch finden.

2.1 Syntax von PL1

Die Syntax einer Logik legt die Datenobjekte fest und definiert damit die Sprache, in der Aussagen formuliert werden können. Folgende Datenobjekte sind in PL1 verfügbar:

Konstantensymbole: „Sokrates“, „5“, „Tisch_1“, „Konstante_3“ …
Abkürzende Bezeichner: a, b, c, d, e.

Variablensymbole: „Menschen“, „Zahlen“, „Tische“, „Datenobjekte“ …
Abkürzende Bezeichner: x, y, z, u, v, w.

Funktionssymbole: „Geburtsjahr“, „+“, „Tischbeine“, „Semantik“ …
Abkürzende Bezeichner: f, g, h, i.

Prädikatensymbole: „ist-Mensch“, „Primzahl“, „ist-Teil-von“, „widersprüchlich“ …
Abkürzende Bezeichner: P, Q, R, S, T.

Aus diesen primitiven Symbolen werden zusammengesetzte Datenobjekte aufgebaut. Dazu ist jedem Funktionssymbol und jedem Prädikatensymbol eine Stelligkeit zugeordnet, die die Anzahl seiner Argumente angibt. Beispielsweise hat „+“ im allgemeinen die Stelligkeit 2. Die Mengen der primitiven Symbole mit ihren Stelligkeiten bilden die sogenannte *Signatur*.

Terme: +(5, Geburtsjahr(Sokrates))
Tischbeine(Tisch_1)
Semantik(Konstante_3)
Abkürzende Bezeichner: r, s, t.

Atome: ist-Mensch(Sokrates)
Primzahl(+(5,5))
ist-Teil-von(Bein_3, Tisch_1)
widersprüchlich(Formel_15)
Abkürzende Bezeichner: L, K, M, N.

Alle Konstanten- und Variablensymbole gelten als Terme; wenn f ein Funktionssymbol der Stelligkeit n ist und t_1 bis t_n Terme sind, dann ist auch $f(t_1,\ldots,t_n)$ ein Term. Wenn P ein Prädikatensymbol der Stelligkeit n ist und t_1 bis t_n Terme sind, dann ist $P(t_1,\ldots,t_n)$ ein Atom. Terme und Atome können der besseren Lesbarkeit halber auch in Infix- oder Outfix-Notation

geschrieben werden. Zum Beispiel ist <(abs(-(x, y)), a) besser lesbar in der Form |x - y| < a. Terme und Atome, die keine Variablensymbole enthalten, bezeichnet man als *Grundterme* und *Grundatome* (englisch *ground terms, ground atoms*).

Atome entsprechen elementaren logischen Aussagen. Terme entsprechen Objekten, über die Aussagen formuliert werden sollen. Komplexere Aussagen werden mit Junktoren und Quantoren aus den elementaren zusammengesetzt.

Junktoren:	$\neg$	nicht (Negation)
	$\wedge$	und (Konjunktion)
	$\vee$	oder (Disjunktion)
	$\Rightarrow$	wenn … dann … (Implikation)
	$\Leftrightarrow$	genau dann wenn (Äquivalenz).
Quantoren:	$\forall$	für alle (Allquantor)
	$\exists$	es gibt (Existenzquantor).
Formeln:	ist-Mensch(Mutter(Sokrates)) $\Leftrightarrow$ ist-Mensch(Sokrates)	
	$\forall$x $\exists$y Primzahl(y) $\wedge$ <(x, y)	
	$\forall$x ist-Tisch(x) $\Rightarrow$ <(1, Anzahl(Tischbeine(x)))	
	Abkürzende Bezeichner: $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$.	

Alle Atome sind Formeln, und wenn $\mathcal{F}$ und $\mathcal{G}$ Formeln sind und x ein Variablensymbol ist, dann sind auch $\neg\mathcal{F}$, $\mathcal{F}\wedge\mathcal{G}$, $\mathcal{F}\vee\mathcal{G}$, $\mathcal{F}\Rightarrow\mathcal{G}$, $\mathcal{F}\Leftrightarrow\mathcal{G}$, $\forall x\mathcal{F}$, $\exists x\mathcal{F}$ Formeln. Um Klammern zu sparen, vereinbart man folgende Bindungspriorität: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\forall$, $\exists$. Quantifizierungen über mehrere Variablen können zusammengezogen werden, z.B. steht $\forall x_1 \ldots x_n\ \mathcal{F}$ für $\forall x_1 \ldots \forall x_n\ \mathcal{F}$.

Einige Sprechweisen sind im Zusammenhang mit Formeln allgemein gebräuchlich. Terme, Atome und Formeln werden oft vereinheitlichend als *Ausdrücke* bezeichnet. In der Formel $\forall x\ \mathcal{F}$ *bindet* der Quantor die Variable x. Die Unterformel $\mathcal{F}$ ist der *Gültigkeitsbereich* (englisch *scope*) der Bindung, ausgenommen Unterformeln von $\mathcal{F}$ der Art $\forall x\ \mathcal{G}$ oder $\exists x\ \mathcal{G}$, in denen die Variable x für $\mathcal{G}$ neu gebunden wird. Je nach Art des bindenden Quantors spricht man auch von *allquantifizierten* oder *existenzquantifizierten* Variablen, ungebundene Variablen heißen *freie* Variablen. Formeln ohne Variablensymbole sind *Grundformeln*. *Aussagenlogische* Formeln enthalten nur nullstellige Prädikatensymbole und damit weder Quantoren noch Terme.

2.2 Semantik von PL1

Die heute übliche Methode, den oben definierten Datenobjekten eine Bedeutung zuzuordnen, geht auf den polnischen Logiker Alfred Tarski zurück, man spricht daher von der *Tarski-semantik* [Tar36]. Ausgangspunkt ist eine nichtleere Menge $\mathcal{U}$, genannt Trägermenge, *Universum* oder *Diskursbereich*. Den syntaktischen Objekten werden durch eine *Interpretation* semantische zugeordnet, und zwar jedem Konstantensymbol ein Element von $\mathcal{U}$, jedem Funktionssymbol eine Abbildung über $\mathcal{U}$ und jedem Prädikatensymbole eine Relationen über $\mathcal{U}$

mit entsprechender Stelligkeit.

Nehmen wir zum Beispiel an, es gibt die beiden Konstantensymbole a und c, ein Funktionssymbol „nächster“, sowie ein Prädikatensymbol „LT“. Dafür betrachten wir zwei verschiedene Interpretationen. Interpretation $\mathcal{J}$ benutzt als Universum die Menge der natürlichen Zahlen und ordnet folgendermaßen zu: dem Konstantensymbol a die Null; c die Zwei; „nächster“ die Nachfolgerabbildung (die jede natürliche Zahl auf die um Eins größere abbildet); „LT“ die arithmetische kleiner-Relation. Interpretation $\mathcal{K}$ benutzt als Universum die Menge {Montag, Dienstag, Mittwoch, Donnerstag, Freitag, Samstag, Sonntag} der Wochentage und ordnet folgendermaßen zu: dem Konstantensymbol a den Sonntag; c den Dienstag; „nächster“ die morgen-Abbildung, die Sonntag auf Montag abbildet, Montag auf Dienstag usw.; „LT“ die kommt-vor-Relation, die sich auf die oben angegebene Auflistung der Tage bezieht (Montag kommt vor allen anderen Wochentagen, Samstag kommt nur vor Sonntag, Sonntag kommt vor keinem anderen Wochentag). Selbstverständlich kann man noch andere Interpretationen über denselben Universen konstruieren.

Unter einer Interpretation läßt sich jeder Grundterm zu einem Element des Universums auswerten. Beispielsweise wird der Term nächster(nächster(c)) unter $\mathcal{J}$ zur natürlichen Zahl Vier und unter $\mathcal{K}$ zum Wochentag Donnerstag ausgewertet. Ein Atom $P(t_1,\ldots,t_n)$ wird entsprechend zu einem Wahrheitswert ausgewertet. Die Auswertung von $t_1,\ldots,t_n$ ergibt ein Tupel von Elementen des Universums, für das man prüft, ob es in der dem Prädikatensymbol P zugeordneten Relation steht. Ist dies der Fall, sagt man, die Interpretation *erfüllt* das Atom oder das Atom ist *wahr* unter der Interpretation. Andernfalls *falsifiziert* die Interpretation das Atom oder das Atom ist *falsch* unter der Interpretation. Beispielsweise erfüllt die Interpretation $\mathcal{J}$ das Atom LT(a, nächster(nächster(c))), da Null kleiner als Vier ist. Dagegen falsifiziert $\mathcal{K}$ dieses Atom, denn nach der obigen Definition kommt Sonntag nicht vor Donnerstag.

Nachdem der Wahrheitswert von Atomen feststeht, kann man ihn auch für beliebige zusammengesetzte Formeln bestimmen. Dies geschieht für die Junktoren mit Hilfe der üblichen Wahrheitstabellen. Eine quantifizierte Formel $\exists x\ \mathcal{F}$ wird von einer Interpretation erfüllt, wenn es eine Zuordnung von x zu einem Element des Universums gibt, für die $\mathcal{F}$ von der Interpretation erfüllt wird. Der Allquantor erfordert entsprechend die Berücksichtigung jeder derartigen Zuordnung. Die Formel $\forall x\, \exists y\, LT(x, y)$ wird also von $\mathcal{J}$ erfüllt, denn zu jeder natürlichen Zahl gibt es tatsächlich eine größere. Dagegen falsifiziert $\mathcal{K}$ die Formel, da der Sonntag in unserer Definition der kommt-vor-Relation das größte Element des Universums ist. Die Formel $\exists xy\, LT(x, y) \wedge \neg LT(x, \text{nächster}(y))$ ist dagegen falsch unter $\mathcal{J}$ und wahr unter $\mathcal{K}$ (z.B. für Samstag und Sonntag), während $\forall x y\, LT(x, y) \Rightarrow \neg LT(y, x)$ von beiden Interpretationen erfüllt wird, weil jede der beiden Relationen eine strikte Ordnung auf ihrem Universum darstellt.

Eine Interpretation, die eine Formel $\mathcal{F}$ erfüllt, heißt auch ein Modell für $\mathcal{F}$. Eine Formel ist

allgemeingültig (*Tautologie*)	wenn sie von jeder Interpretation erfüllt wird;
erfüllbar	wenn sie von wenigstens einer Interpretation erfüllt wird;
falsifizierbar	wenn sie von wenigstens einer Interpretation falsifiziert wird;
unerfüllbar (*widersprüchlich*)	wenn sie von jeder Interpretation falsifiziert wird.

Offensichtlich ist eine Formel genau dann allgemeingültig, wenn sie nicht falsifizierbar ist, und genau dann unerfüllbar, wenn sie nicht erfüllbar ist. In PL1 gilt darüberhinaus die wichtige Eigenschaft, daß eine Formel genau dann allgemeingültig ist, wenn ihre Negation unerfüllbar ist, und genau dann erfüllbar ist, wenn ihre Negation falsifizierbar ist.

Die Formel $\forall x \exists y \, LT(x, y)$ wird von $\mathcal{I}$ erfüllt und von $\mathcal{K}$ falsifiziert, also ist sie sowohl erfüllbar als auch falsifizierbar, somit weder unerfüllbar noch allgemeingültig. Die Formeln $P \Rightarrow Q \;\Leftrightarrow \neg P \vee Q$ und $P \vee \neg P$ sind beide allgemeingültig, also auch erfüllbar. Die Formel $P \wedge \neg P$ ist unerfüllbar und damit auch falsifizierbar.

Auf ein berühmtes Theorem von Leopold Löwenheim und Thoralf Skolem geht die Sichtweise zurück, die Menge der Grundterme selbst als ausgezeichnetes Universum aufzufassen. Derselbe Grundgedanke erscheint auch in einer späteren Arbeit des französischen Philosophen Jacques Herbrand. Aufgrund eines historischen Irrtums hat sich im Bereich der Deduktionssysteme für dieses ausgezeichnete Universum die Bezeichnung *Herbranduniversum* eingebürgert. Eine *Herbrandinterpretation* ordnet jedem Konstantensymbol sich selbst zu und jedem Funktionssymbol f mit der Stelligkeit n eine termerzeugende Abbildung, die n Grundterme $t_1, \ldots, t_n$ als Argumente nimmt und auf den Grundterm $f(t_1, \ldots, t_n)$ abbildet. Jedes Prädikatensymbol P der Stelligkeit m wird einer m-stelligen Relation über Grundtermen zugeordnet, indem ganz konkret die Menge der m-Tupel $(t_1, \ldots, t_m)$ von Grundtermen angegeben wird, für die die Interpretation das Atom $P(t_1, \ldots, t_m)$ erfüllt.

Zu jeder beliebigen Interpretation einer Formel gibt es auch eine Herbrandinterpretation, unter der die Formel denselben Wahrheitswert hat. Jede erfüllbare Formel besitzt demnach ein sogenanntes *Herbrandmodell*.[1] Für viele Untersuchungen genügt es daher, nur Herbrandinterpretationen und -modelle zu betrachten.

Der semantische Begriff des *Folgerns* baut nun auf dem Modellbegriff auf: Eine Formel $\mathcal{G}$ *folgt* aus einer Formel $\mathcal{F}$ (geschrieben $\mathcal{F} \models \mathcal{G}$), wenn alle Modelle von $\mathcal{F}$ auch Modelle von $\mathcal{G}$ sind (aber nicht notwendigerweise umgekehrt). Es gelten zum Beispiel $\forall x \, P(x) \models P(a)$ und $Q \wedge R \models Q$, nicht aber $Q \vee R \models Q$. Da die Gültigkeit von $\mathcal{F} \models \mathcal{G}$ von allen denkbaren Modellen für $\mathcal{F}$ abhängt, kommt diese Definition für eine algorithmische Überprüfung offensichtlich nicht in Frage.

Durch einfaches Umformen der Definitionen erhält man einen wichtigen Zusammenhang zwischen der semantischen Folgerungsbeziehung und dem Implikationsjunktor: Für alle Formeln $\mathcal{F}$, $\mathcal{G}$ ohne freie Variablen gilt $\mathcal{F} \models \mathcal{G}$ genau dann, wenn die Formel $\mathcal{F} \Rightarrow \mathcal{G}$ allgemeingültig ist. Dieser Satz erlaubt die Rückführung von Folgerungen auf das Problem, die Allgemeingültigkeit einer Formel nachzuweisen.

---

[1] Das ist nicht ganz korrekt für Formeln wie $\exists x \exists y \, P(x,a) \wedge \neg P(y,a)$, deren Pränexform Existenzquantoren enthält. Allerdings kann jede Formel so transformiert werden, daß diese Existenzquantoren verschwinden – siehe den nächsten Abschnitt.

2.3 Normalformen in PL1

Prädikatenlogische Formeln können beliebig verschachtelt sein. Damit die Operationen auf Formeln nicht zu kompliziert werden, versucht man häufig, die Formeln zunächst in eine möglichst einfache standardisierte Form zu bringen. Eine Handhabe dazu bieten allgemeingültige Formeln der Art $\mathcal{A} \Leftrightarrow \mathcal{B}$, die modellerhaltende Umformungen erlauben. Man kann nämlich nach obigem Satz die Unterformeln $\mathcal{A}$ und $\mathcal{B}$ in anderen Formeln beliebig gegeneinander austauschen, ohne den Wahrheitswert für irgendeine Interpretation zu verändern. Die wichtigsten tautologischen Äquivalenzen sind:

$\mathcal{F} \Leftrightarrow \mathcal{G}$	$\Leftrightarrow$	$\mathcal{F} \Rightarrow \mathcal{G} \wedge \mathcal{G} \Rightarrow \mathcal{F}$	(Hiermit läßt sich $\Leftrightarrow$ völlig eliminieren)
$\mathcal{F} \Rightarrow \mathcal{G}$	$\Leftrightarrow$	$\neg\, \mathcal{F} \vee \mathcal{G}$	(Hiermit läßt sich $\Rightarrow$ völlig eliminieren)
$\neg(\mathcal{F} \wedge \mathcal{G})$	$\Leftrightarrow$	$\neg\, \mathcal{F} \vee \neg\mathcal{G}$	
$\neg(\mathcal{F} \vee \mathcal{G})$	$\Leftrightarrow$	$\neg\, \mathcal{F} \wedge \neg\mathcal{G}$	
$\neg\forall x\, \mathcal{F}$	$\Leftrightarrow$	$\exists x \neg \mathcal{F}$	
$\neg\exists x\, \mathcal{F}$	$\Leftrightarrow$	$\forall x \neg \mathcal{F}$	
$(\forall x \mathcal{F}) \wedge \mathcal{G}$	$\Leftrightarrow$	$\forall x(\mathcal{F} \wedge \mathcal{G})$	sofern x nicht frei in $\mathcal{G}$ vorkommt
$(\forall x \mathcal{F}) \vee \mathcal{G}$	$\Leftrightarrow$	$\forall x(\mathcal{F} \vee \mathcal{G})$	sofern x nicht frei in $\mathcal{G}$ vorkommt
$(\exists x \mathcal{F}) \wedge \mathcal{G}$	$\Leftrightarrow$	$\exists x(\mathcal{F} \wedge \mathcal{G})$	sofern x nicht frei in $\mathcal{G}$ vorkommt
$(\exists x \mathcal{F}) \vee \mathcal{G}$	$\Leftrightarrow$	$\exists x(\mathcal{F} \vee \mathcal{G})$	sofern x nicht frei in $\mathcal{G}$ vorkommt
$\forall x \mathcal{F} \wedge \forall x \mathcal{G}$	$\Leftrightarrow$	$\forall x(\mathcal{F} \wedge \mathcal{G})$	
$\exists x \mathcal{F} \vee \exists x \mathcal{G}$	$\Leftrightarrow$	$\exists x(\mathcal{F} \vee \mathcal{G})$	
$\mathcal{F} \vee (\mathcal{G} \wedge \mathcal{H})$	$\Leftrightarrow$	$(\mathcal{F} \vee \mathcal{G}) \wedge (\mathcal{F} \vee \mathcal{H})$	(Distributivität)
$\mathcal{F} \wedge (\mathcal{G} \vee \mathcal{H})$	$\Leftrightarrow$	$(\mathcal{F} \wedge \mathcal{G}) \vee (\mathcal{F} \wedge \mathcal{H})$	(Distributivität)

Mit Hilfe dieser Transformationen (und einer Variablenumbenennungsregel, mit der man z. B. das x in der linken Unterformel von $(\forall x \mathcal{F}) \wedge \mathcal{G}$ durch ein x' ersetzen kann, das nicht frei in $\mathcal{G}$ vorkommt), läßt sich eine Formel in eine äquivalente *Pränexform* überführen, in der alle Quantoren nach außen vor den quantorfreien *Kern* gezogen sind. Zur weiteren Vereinfachung dient die nach dem norwegischen Mathematiker Thoralf Skolem benannte *Skolemisierung* [Sko20]: Eine Formel $\forall x_1 \ldots x_n \exists y\ \mathcal{F}$ in Pränexform wird umgeformt in $\forall x_1 \ldots x_n\ \mathcal{F}^*$, wobei $\mathcal{F}^*$ aus $\mathcal{F}$ durch Ersetzen aller freien Vorkommnisse von y durch den Term $f_y(x_1,\ldots,x_n)$ gebildet wird. Dabei ist f_y ein neues Funktionssymbol, eine sogenannte Skolemfunktion. Der Term $f_y(x_1,\ldots,x_n)$ repräsentiert gerade ein y, das zu gegebenen $x_1,\ldots,x_n$ existieren soll. Die Skolemisierung ist keine modellerhaltende Umformung, eine Formel ist also nicht äquivalent zu ihrer skolemisierten Form. Sie ist aber immerhin genau dann erfüllbar bzw. unerfüllbar, wenn dies auch für die skolemisierte Form gilt (eine dazu duale Skolemisierung der Allquantoren würde entsprechend die Allgemeingültigkeit bzw. Falsifizierbarkeit erhalten). Da nach der Skolemisierung nur noch Allquantoren vorkommen, kann man den Quantorpräfix auch ganz weglassen und freie Variable im Kern implizit als allquantifiziert betrachten.

Durch weitere Anwendungen von modellerhaltenden Transformationen ist es möglich, im Kern einer Formel in Pränexform die Junktoren $\Leftrightarrow$ und $\Rightarrow$ zu eliminieren sowie alle Negationszeichen nach innen bis unmittelbar vor die Atome zu ziehen. Atome und negierte Atome heißen auch *Literale*. Der verbleibende Kern ist also nur noch aus Literalen und beliebig

verschachtelten Konjunktionen und Disjunktionen aufgebaut. Ist zusätzlich der Quantorpräfix durch Skolemisierung beseitigt, spricht man von der *Negationsnormalform* (englisch *negation normal form*).

Schließlich ermöglichen die beiden Distributivgesetze je nach Wunsch die Erzeugung der *konjunktiven Normalform* ($(L_1 \vee \ldots \vee L_n) \wedge \ldots \wedge (K_1 \vee \ldots \vee K_m)$) oder der (seltener benutzten) *disjunktiven Normalform* ($(L_1 \wedge \ldots \wedge L_n) \vee \ldots \vee (K_1 \wedge \ldots \wedge K_m)$), wobei die L_i und K_i Literale sind. Da die beiden Junktoren $\wedge$ und $\vee$ assoziativ, kommutativ und idempotent sind, also gerade die Eigenschaften von Mengenkonstruktoren haben, werden diese Formeln üblicherweise auch als Mengen von Mengen von Literalen geschrieben: $\{\{L_1, \ldots, L_n\}, \ldots, \{K_1, \ldots, K_m\}\}$.

Die Mengendarstellung der konjunktiven Normalform heißt auch *Klauselform*. Die inneren Mengen $\{L_1, \ldots, L_n\}$ werden als *Klauseln* bezeichnet, die gesamte Menge von Klauseln als *Klauselmenge*. Klauseln, die nur ein einziges Literal enthalten, heißen *unäre* Klauseln (englisch *unit clauses*). Jede Formel läßt sich also in eine Klauselmenge umwandeln, die genau dann erfüllbar bzw. unerfüllbar ist, wenn dies auch für die Ausgangsformel gilt. Eine wichtige Rolle spielen Klauseln, die nicht mehr als ein positives Literal haben. Sie entstehen unter anderem durch die Normalisierung von Implikationen der Art $L_1 \wedge \ldots \wedge L_n \Rightarrow L_{n+1}$ und heißen *Hornklauseln*. Hornklauselmengen haben gewisse angenehme Eigenschaften, die z.B. in der Programmiersprache PROLOG ausgenutzt werden.

Beispiel: Umwandlung einer Formel in die verschiedenen Normalformen

Ausgangsformel:	$\forall\varepsilon\,(\varepsilon{>}0 \Rightarrow \exists\delta\,(\delta{>}0 \wedge \forall xy\,(\lvert x{-}y\rvert{<}\delta \Rightarrow \lvert g(x){-}g(y)\rvert{<}\varepsilon)))$
Pränexform:	$\forall\varepsilon\,\exists\delta\,\forall xy\quad \varepsilon{>}0 \Rightarrow \delta{>}0 \wedge (\lvert x{-}y\rvert{<}\delta \Rightarrow \lvert g(x){-}g(y)\rvert{<}\varepsilon)$
Skolemisierte Pränexform:	$\varepsilon{>}0 \Rightarrow f_\delta(\varepsilon){>}0 \wedge (\lvert x{-}y\rvert{<}f_\delta(\varepsilon) \Rightarrow \lvert g(x){-}g(y)\rvert{<}\varepsilon)$
Negationsnormalform:	$\neg\varepsilon{>}0 \vee (f_\delta(\varepsilon){>}0 \wedge (\neg\lvert x{-}y\rvert{<}f_\delta(\varepsilon) \vee \lvert g(x){-}g(y)\rvert{<}\varepsilon))$
Disjunktive Normalform:	$(\neg\varepsilon{>}0) \vee (f_\delta(\varepsilon){>}0 \wedge \neg\lvert x{-}y\rvert{<}f_\delta(\varepsilon)) \vee (f_\delta(\varepsilon){>}0 \wedge \lvert g(x){-}g(y)\rvert{<}\varepsilon)$
Konjunktive Normalform:	$(\neg\varepsilon{>}0 \vee f_\delta(\varepsilon){>}0) \wedge (\neg\varepsilon{>}0 \vee \neg\lvert x{-}y\rvert{<}f_\delta(\varepsilon) \vee \lvert g(x){-}g(y)\rvert{<}\varepsilon)$
Klauselform:	$\{\{\neg\varepsilon{>}0,\ f_\delta(\varepsilon){>}0\},\quad \{\neg\varepsilon{>}0,\ \neg\lvert x{-}y\rvert{<}f_\delta(\varepsilon),\ \lvert g(x){-}g(y)\rvert{<}\varepsilon\}\}.$

Die Ausgangsformel ist erfüllbar, sie definiert die gleichmäßige Stetigkeit einer Funktion g. Also ist auch die Klauselmenge am Ende erfüllbar. ■

Durch die Elimination des Äquivalenzzeichens und die Anwendung der Distributivgesetze wird eine Vervielfältigung von Teilformeln erzwungen. Abhängig von der Schachtelungstiefe der Formel kann dies zu einer exponentiellen Zunahme von Unterformeln führen. Um diesen Effekt soweit abzumildern, daß die Zunahme nur linear ist, läßt sich ein Trick anwenden, dessen Grundidee an folgendem Beispiel erläutert wird: Angenommen die Formel $\mathcal{F} \vee (\mathcal{G} \wedge \mathcal{H})$ soll in konjunktive Normalform gebracht werden. Die Anwendung des Distributivgesetzes würde die Teilformel $\mathcal{F}$ verdoppeln. Stattdessen wird ein neues Prädikatensymbol P eingeführt, das als Abkürzung für die Teilformel $(\mathcal{G} \wedge \mathcal{H})$ dienen soll. Aus der Formel $\mathcal{F} \vee (\mathcal{G} \wedge \mathcal{H})$ wird dann $(\mathcal{F} \vee P(x_1,\ldots,x_n)) \wedge (\neg P(x_1,\ldots,x_n) \vee \mathcal{G}) \wedge (\neg P(x_1,\ldots,x_n) \vee \mathcal{H})$, wobei $x_1,\ldots,x_n$ die freien Variablen von $(\mathcal{G} \wedge \mathcal{H})$ sind. $P(x_1,\ldots,x_n)$ kommt nun zwar mehrfach vor, es ist aber nur ein Atom und

muß daher nicht mehr weiter bearbeitet werden. Die möglicherweise komplexe Formel $\mathcal{F}$ kommt dagegen weiterhin nur einmal vor, und darin liegt der Spareffekt.

2.4 Beschränkungen und Modifikationen von PL1

PL1 erlaubt nur die Quantifizierung über Objektvariable, nicht jedoch die Quantifizierung über Funktions- und Prädikatenvariable. Eine Formel „$\forall$P Symmetrisch(P) $\Leftrightarrow$ ($\forall$xy P(x,y) $\Leftrightarrow$ P(y,x))“ ist daher erst in Prädikatenlogik zweiter Stufe (PL2) zulässig, die die Quantifizierung über Prädikaten- und Funktionsvariable erlaubt. Da ein Prädikat mit der Menge von Objekten identifiziert werden kann, für die es zutrifft, kann in PL2 im Grunde sowohl über Objekte als auch über Mengen von Objekten quantifiziert werden. In der dritten Stufe ist dann zusätzlich die Quantifizierung über Mengen von Prädikaten- und Funktionsvariablen und damit über Mengen von Mengen von Objekten erlaubt, und analog setzen sich die höheren Stufen fort. In PL1 ist all das nicht zugelassen. Strukturen, bei deren Axiomatisierung es darauf ankommt, gleichzeitig über Objekte und Mengen von Objekten zu quantifizieren, können daher in PL1 nicht mehr hinreichend beschrieben werden.

Ein bekanntes Beispiel für eine solche Struktur ist die Menge der reellen Zahlen, die erst durch das Supremumsaxiom ausreichend (d.h. bis auf Isomorphie) charakterisiert wird. Dieses besagt, daß es für jede nach oben beschränkte *Teilmenge* von reellen Zahlen eine kleinste obere Schranke gibt, und das ist gerade eine Aussage der kritischen Art. Der Übergang von abzählbaren zu überabzählbaren Modellen scheint also eine Schranke für PL1 darzustellen. In der Tat gilt nach dem Theorem von Löwenheim und Skolem, daß jede aus endlich vielen Zeichen aufgebaute erfüllbare PL1-Formel ein endliches oder abzählbares Modell hat. Damit ist eine eindeutige Charakterisierung überabzählbarer Strukturen ausgeschlossen. Das heißt jedoch nicht, daß man überhaupt keine interessanten Eigenschaften davon in PL1 formulieren und beweisen könnte.

Kann man in PL1 zumindest alle abzählbaren Strukturen bis auf Isomorphie eindeutig axiomatisieren? Leider ist auch das nicht der Fall (jedenfalls nicht mit endlich vielen Axiomen), wie das Beispiel der natürlichen Zahlen zeigt. Die natürlichen Zahlen werden durch die Peanoaxiome beschrieben, deren in PL1 formulierbarer Teil besagt, daß es eine Basiskonstante 0 gibt und eine injektive Nachfolgerfunktion, so daß jede natürliche Zahl außer der 0 Nachfolger von genau einer anderen ist. Die intuitiven natürlichen Zahlen erfüllen diese Axiome, man kann aber auch größere Modelle konstruieren, indem man zu den „echten“ natürlichen Zahlen beispielsweise die Wochentage mit der naheliegenden Erweiterung der Nachfolgerfunktion hinzunimmt. Um solche Modelle auszuschließen, benötigt man das erst in PL2 formulierbare Induktionsaxiom „$\forall$P P(0) $\wedge$ ($\forall$x P(x) $\Rightarrow$ P(x+1)) $\Rightarrow$ $\forall$x P(x)“. Beweise, die von diesem Axiom abhängen, sind eben Induktionsbeweise und können ohne Zusatzmechanismen nicht in Kalkülen geführt werden, die nur auf PL1 aufbauen. Dies macht vielleicht deutlich, daß für eine einigermaßen effiziente Behandlung von Formeln höherer Stufe andere Methoden nötig sind, als sie in Deduktionssystemen für PL1 zur Verfügung stehen. Logiken höherer Stufe sind nicht ohne weiteres algorithmisierbar, da viele für PL1 noch geltende grundlegende Eigenschaften verloren gehen. Daher beschränken wir uns im

wesentlichen auf PL1.

Es gibt syntaktische Varianten von PL1, die zwar nicht die prinzipielle Ausdrucksfähigkeit erhöhen, aber eine kompaktere Darstellung von Formeln erlauben. Damit ist in den einzelnen Unterausdrücken mehr Information konzentriert, die dann von Kalkülen ausgenutzt werden kann, um sinnlose Aussagen zu erkennen und zu vermeiden. Die wichtigsten derartigen Varianten sind die *mehrsortigen* Logiken. Die Grundidee dabei ist, nicht einfach ein strukturloses Universum zu betrachten, sondern die Objekte in Klassen einzuteilen. Aussagen werden nur für Objekte bestimmter Klassen formuliert und sind für andere Objekte gar nicht definiert. Beispielsweise könnte man Aussagen über Menschen und über natürliche Zahlen in einer mehrsortigen Logik so formulieren, daß die Behauptung, Sokrates und Napoleon seien teilerfremd, nicht einfach wahr oder falsch, sondern syntaktisch unzulässig wäre.

Dazu stellen mehrsortige Logiken zusätzlich zu den genannten Datenobjekten noch eine Menge von *Sorten* zur Verfügung. Die Sorten können einfach eine unstrukturierte Menge bilden (*flache* Sortenstruktur) oder durch eine partielle Ordnung $\sqsubset$, die *Untersortenbeziehung*, geordnet sein (*hierarchische* Sortenstruktur). Man würde etwa die Menge der Sortensymbole {Integer, Menschen} flach und die Menge {Integer, Real, Complex} hierarchisch strukturieren. Die zulässige Struktur der Untersortenbeziehung, z.B. linear, baumförmig, halbverbandsartig oder verbandsartig, hat Einfluß auf die möglichen Einsetzungen für Variablensymbole und beeinflußt daher die Konstruktion von Kalkülen für die jeweilige spezielle Logik.

Die Signatur der Logik wird nun folgendermaßen erweitert: jedem Konstanten- und Variablensymbol ist eine Sorte zugeordnet, Funktionssymbolen werden Argument/Ergebnissorten-Beziehungen mit der entsprechenden Stelligkeit zugeordnet, Prädikatensymbolen die zulässigen Argumentsorten. Beispielsweise spezifiziert Geburtsjahr: Menschen $\rightarrow$ Integer eine Funktion, die nur Argumente der Sorte Menschen zuläßt und dann ein Ergebnis der Sorte Integer liefert. Mit +: (Real $\times$ Real $\rightarrow$ Real, Real $\times$ Integer $\rightarrow$ Real, Integer $\times$ Real $\rightarrow$ Real, Integer $\times$ Integer $\rightarrow$ Integer) charakterisiert man mehrere mögliche Beziehungen der Argumentsorten zu den Ergebnissorten der Funktion +. Teilerfremd: Integer $\times$ Integer besagt, daß das Prädikatensymbol Teilerfremd nur Terme vom Typ Integer als Argument akzeptiert.

Die Sorte eines zusammengesetzten Terms wird nun aus der Sortendeklaration für das oberste Funktionssymbol und eventuell aus den Sorten der Unterterme berechnet. Ein korrekter Term Geburtsjahr(…) hat zum Beispiel immer die Sorte Integer, während die Sorte eines Terms +(…,…) von den Sorten der Argumente abhängt. Ein Term wie Geburtsjahr(+(…,…)) ist nicht *sortenrecht* und daher verboten. Damit wird rein syntaktisch die Formulierung vieler unsinniger Aussagen verhindert.

Daß mehrsortige Logiken nicht wirklich ausdrucksstärker sind als unsortierte, zeigt sich darin, daß die gesamte Information über die Sorten in einstellige Prädikate kodiert werden kann: Die Untersortenbeziehungen $S \sqsubset P$ werden als Implikationen ausgedrückt: $\forall x\, S(x) \Rightarrow P(x)$. Quantifizierungen über sortierte Variable $\forall x{:}S\ \mathcal{F}$ werden in die Form $\forall x\, S(x) \Rightarrow \mathcal{F}$ übersetzt, und die Argument/Ergebnissorten-Beziehungen der Funktionen $f: S_1 \times \ldots \times S_n \rightarrow S$ lassen sich durch Formeln $\forall x_1 \ldots x_n\ S_1(x_1) \wedge \ldots \wedge S_n(x_n) \Rightarrow S(f(x_1,\ldots,x_n))$ beschreiben. Semantisch unsinnige Terme wie der oben erwähnte Term Geburtsjahr(+(…,…)) sind in dieser Form

jedoch nicht ausgeschlossen.

Mehrsortige Logiken und Kalküle dafür wurden von Jacques Herbrand [Her30b], Arnold Oberschelp [Obe62], Christoph Walther [Wal87], Manfred Schmidt-Schauß [Sch85] und einigen anderen entwickelt. Die noch recht einfache Idee der Sorten ist in der intuitionistischen Typentheorie (höherer Stufe) von Per Martin-Löf [Mar84] extrem verallgemeinert worden. Dort wird alle Information in Form von Typen kodiert und manipuliert, und andere Terme treten überhaupt nicht mehr auf.

Andere Varianten von PL1 erhält man durch Vorgabe von bestimmten Symbolen mit einer festen Bedeutung. Am gebräuchlichsten ist ein spezielles zweistelliges Prädikatensymbol (etwa $\equiv$) als fester Bestandteil der Syntax, das für die Gleichheitsrelation steht. Man kann damit Atome der Form $\equiv(s, t)$ (oder $s \equiv t$ in Infixschreibweise) bilden. Semantische Begriffe werden dann so abgewandelt, daß jede Interpretation für dieses Prädikatensymbol automatisch die Eigenschaften der Gleichheit sicherstellt. Sofern Verwechslungen ausgeschlossen sind, schreibt man statt $\equiv(s,t)$ oft auch einfach $s = t$, das heißt, man benutzt das Gleichheitszeichen der Metaebene auch als Prädikatensymbol.

3 Kalküle für die Prädikatenlogik erster Stufe

Ein Kalkül erweitert eine Logik um den syntaktischen Begriff des Ableitens, der den semantischen Folgerungsbegriff nachbilden soll. Es werden also Syntax und Semantik einer Logik vorausgesetzt, möglicherweise eingeschränkt auf eine Teilmenge von Formeln (etwa solche in einer Normalform). Wie in Abschnitt 2.2 gezeigt wurde, kann die semantische Folgerung auf die Überprüfung der Allgemeingültigkeit einer Formel zurückgeführt werden. Dazu definiert ein Kalkül zunächst eine Menge von *logischen Axiomen*, die eine Minimalausstattung von allgemeingültigen Formeln darstellen, die elementaren Tautologien. In diesem Fall spricht man von einem *positiven* Kalkül. Dual dazu kann man auch Kalküle konstruieren, deren logische Axiome unerfüllbar sind und somit den elementaren Widersprüchen entsprechen. Dann spricht man von einem *negativen* Kalkül. Da eine Formel genau dann allgemeingültig ist, wenn ihre Negation unerfüllbar ist, sind beide Arten von Kalkülen völlig gleichwertig.

Zusätzlich zu den logischen Axiomen stellt ein Kalkül eine Menge von *Schlußregeln* zur Verfügung, mit denen aus Formeln weitere Formeln abgeleitet werden können. Diese Regeln sind als rein syntaktische Symbolmanipulation definiert, die keine Kenntnis der Bedeutung der Formeln erfordern. Eine Schlußregel hat die Form

$$\frac{\mathcal{F}_1 \quad \cdots \quad \mathcal{F}_n}{\mathcal{F}}$$

und erlaubt, aus den Formeln $\mathcal{F}_1, \ldots, \mathcal{F}_n$ die neue Formel $\mathcal{F}$ abzuleiten. Ist $\mathcal{F}$ durch beliebig viele aufeinanderfolgende Anwendungen von Schlußregeln aus $\mathcal{F}_1, \ldots, \mathcal{F}_n$ ableitbar, schreibt man $\mathcal{F}_1, \ldots, \mathcal{F}_n \vdash \mathcal{F}$. Prinzipiell sind zwei Vorgehensweisen möglich. Man kann erstens die Schlußregeln ausgehend von den logischen Axiomen so lange anwenden, bis die zu

beweisende (je nach Art des Kalküls allgemeingültige oder unerfüllbare) Formel abgeleitet ist. Einen Kalkül mit dazu geeigneten Schlußregeln nennt man einen *deduktiven Kalkül*. Zweitens kann man die Schlußregeln auch ausgehend von der zu beweisenden Formel so lange anwenden, bis man sie auf logische Axiome zurückgeführt hat. In diesem Fall spricht man von einem *Testkalkül* [Ric78].

Man kann einen deduktiven Kalkül in einen Testkalkül umwandeln und umgekehrt, indem man die Richtung der Schlußregeln einfach umdreht. Dies kann aber drastische Auswirkungen auf die Verzweigungsrate des Suchraums haben. Testkalküle verhalten sich zu deduktiven Kalkülen ähnlich wie rückwarts verkettende zu vorwärts verkettenden Zustandsübergangssystemen in der Künstlichen Intelligenz.

Generell gilt, daß umso weniger logische Axiome gebraucht werden, je stärker die Schlußregeln sind. Manche Kalküle kommen sogar ganz ohne logische Axiome aus.

Zur Beurteilung eines Kalküls spielen die Begriffe *Korrektheit* (englisch *soundness*) und Vollständigkeit (englisch *completeness*) eine zentrale Rolle. Ein positiver deduktiver Kalkül ist korrekt, wenn alle logischen Axiome allgemeingültig sind und ableitbare Formeln stets semantisch folgen. Wann immer $\mathcal{F}_1, \ldots, \mathcal{F}_n \vdash \mathcal{F}$ gilt, muß also auch $\mathcal{F}_1 \wedge \ldots \wedge \mathcal{F}_n \models \mathcal{F}$ gelten (und damit, nach dem Satz in Abschnitt 2.2, die Formel $\mathcal{F}_1 \wedge \ldots \wedge \mathcal{F}_n \Rightarrow \mathcal{F}$ allgemeingültig sein). Offensichtlich reicht es aus, wenn jede einzelne Schlußregel diese Bedingung erfüllt. Ein positiver deduktiver Kalkül ist vollständig, wenn jede allgemeingültige Formel durch endlich viele Anwendungen der Schlußregeln aus den logischen Axiomen abgeleitet werden kann.

Für die anderen Klassen von Kalkülen paßt man die Definitionen entsprechend an. Ein Zusammenhang zwischen den vier Klassen ist dabei bedeutsam. Wenn mit den Regeln eines korrekten positiven deduktiven Kalküls oder eines korrekten negativen Testkalküls aus einer Formel $\mathcal{F}$ eine Formel $\mathcal{G}$ abgeleitet werden kann, dann folgt $\mathcal{G}$ semantisch aus $\mathcal{F}$. Bei den beiden anderen Kombinationen ist es gerade umgekehrt. Positive deduktive Kalküle und negative Testkalküle arbeiten also in dieselbe Richtung, das heißt, beide leiten semantische Konsequenzen aus Formeln ab.

Die Art der Logik legt fest, ob Kalküle mit bestimmten Eigenschaften möglich sind. Die wichtigsten Resultate zu diesem Problemkreis, die zum größten Teil in der ersten Hälfte dieses Jahrhunderts gefunden wurden, sind:

Aussagenlogik ist entscheidbar. Die bekannte Wahrheitstabellenmethode ist beispielsweise geeignet, für jede aussagenlogische Formel die Allgemeingültigkeit, Erfüllbarkeit, Falsifizierbarkeit oder Unerfüllbarkeit nachzuweisen.

PL1 ist nicht entscheidbar [Chu36], aber noch vollständig [Göd30], d.h. es gibt vollständige positive deduktive Kalküle für PL1. Die Menge der allgemeingültigen Formeln von PL1 ist rekursiv aufzählbar, die komplementäre Menge der falsifizierbaren Formeln aber nicht (ebenso sind die unerfüllbaren Formeln rekursiv aufzählbar, die erfüllbaren aber nicht). In der Praxis bedeutet dies, daß das Beweisproblem „Voraussetzungen $\models$ Behauptung“ nur semientscheidbar ist: folgt die Behauptung wirklich, läßt sich das mit einem Kalkül in endlicher Zeit nachweisen, andernfalls terminiert der Beweisversuch im allgemeinen nicht. PL2 und alle höherstufigen

Logiken sind nicht mehr vollständig. Es gibt Formeln, deren Allgemeingültigkeit mit keinem denkbaren Kalkül mehr nachgewiesen werden kann [Göd31]. Dies sind die Aussagen der berühmten Gödelschen Vollständigkeits- und Unvollständigkeitssätze.

Nach einem Satz von Per Lindström [Lin69] ist PL1 überhaupt die ausdrucksstärkste vollständige Logik. Das heißt, jede Erweiterung von PL1, die zum Beispiel eingeschränkte Quantifizierung über Funktions- und Prädikatenvariable erlaubt, ist entweder nicht wirklich ausdrucksstärker und daher in PL1 simulierbar, oder es gibt keinen vollständigen Kalkül mehr dafür (vgl. [EFT78]).

3.1 Gentzen-Kalküle (Kalküle des natürlichen Schließens)

Der von dem deutschen Logiker Gerhard Gentzen in seiner Dissertation entwickelte Kalkül ist ein positiver deduktiver Kalkül [Gen35]. Es gibt ihn in verschiedenen Varianten, für die intuitionistische Prädikatenlogik, für die klassische Prädikatenlogik erster Stufe und jeweils in einer organisatorischen Abwandlung, bekannt als Sequenzenkalkül. Gentzens Hauptanliegen war die Nachbildung der tatsächlich in der Mathematik gebräuchlichen Schlußweisen durch syntaktische Schlußregeln. Seine Kalküle sind auch deshalb von besonderem Interesse, weil sich ihre Grundidee für die Entwicklung von Kalkülen für nichtklassische Logiken sehr gut bewährt hat.

Der einfachste Kalkül, NJ, hat 13 Schlußregeln und benötigt keine logischen Axiome. Die 13 Schlußregeln in der von Gentzen verwendeten Notation sind:

$$\frac{\mathcal{F}\quad\mathcal{G}}{\mathcal{F}\wedge\mathcal{G}}\ \text{UE} \qquad \frac{\mathcal{F}}{\mathcal{F}\vee\mathcal{G}}\ \frac{\mathcal{G}}{\mathcal{F}\vee\mathcal{G}}\ \text{OE} \qquad \frac{\begin{matrix}[\mathcal{F}]\\ \mathcal{G}\end{matrix}}{\mathcal{F}\Rightarrow\mathcal{G}}\ \text{FE} \qquad \frac{\mathcal{F}a}{\forall x\,\mathcal{F}x}\ \text{AE} \qquad \frac{\mathcal{F}a}{\exists x\,\mathcal{F}x}\ \text{EE} \qquad \frac{\begin{matrix}[\mathcal{F}]\\ \square\end{matrix}}{\neg\mathcal{F}}\ \text{NE}$$

$$\frac{\mathcal{F}\wedge\mathcal{G}}{\mathcal{F}}\ \frac{\mathcal{F}\wedge\mathcal{G}}{\mathcal{G}}\ \text{UB} \qquad \frac{\mathcal{F}\vee\mathcal{G}\quad\begin{matrix}[\mathcal{F}]\ \mathcal{H}\end{matrix}\quad\begin{matrix}[\mathcal{G}]\\ \mathcal{H}\end{matrix}}{\mathcal{H}}\ \text{OB} \qquad \frac{\mathcal{F}\quad\mathcal{F}\Rightarrow\mathcal{G}}{\mathcal{G}}\ \text{FB} \qquad \frac{\forall x\,\mathcal{F}x}{\mathcal{F}a}\ \text{AB} \qquad \frac{\exists x\,\mathcal{F}x\quad\begin{matrix}[\mathcal{F}a]\\ \mathcal{H}\end{matrix}}{\mathcal{H}}\ \text{EB} \qquad \frac{\mathcal{F}\quad\neg\mathcal{F}}{\square}\ \text{NB} \qquad \frac{\square}{\mathcal{F}}$$

Die oberen sechs Schlußregeln führen logische Zeichen ein. Die UE-Regel („Und-Einführung“) erlaubt beispielsweise, aus zwei Formeln eine neue abzuleiten, die durch Konjunktion der beiden gebildet wird. Die OE-Regel („Oder-Einführung“) leitet aus einer Formel die Disjunktion der gegebenen mit einer beliebigen anderen Formel ab. Entsprechend beseitigen die unteren Regeln logische Zeichen. Mit der UB-Regel („Und-Beseitigung“) kann man etwa aus einer Konjunktion eine der beiden Teilformeln ableiten. Die in eckigen Klammern eingeschlossenen Formeln stehen für Annahmen, aus denen die darunterstehende Formel abgeleitet wurde. Die OB-Regel („Oder-Beseitigung“) bedeutet: Wenn die Formel $\mathcal{F}\vee\mathcal{G}$ abgeleitet wurde, außerdem aus der Annahme $\mathcal{F}$ die Formel $\mathcal{H}$ sowie unabhängig davon aus der Annahme $\mathcal{G}$ ebenfalls die Formel $\mathcal{H}$ abgeleitet wurde, dann darf $\mathcal{H}$ als neue Formel hinzugefügt werden, die nicht mehr von den Annahmen $\mathcal{F}$ und $\mathcal{G}$ abhängt (man muß also die Abhängigkeiten der Formeln

untereinander verwalten). Die Schlußregel FB („Folgt-Beseitigung“) ist bekannter unter dem Namen *Modus Ponens*.

Bei der AE-Regel („All-Einführung“) ist die Nebenbedingung zu beachten, daß $\mathcal{F}a$ von keiner Annahme abhängen darf, in der a vorkommt. Eine ähnliche Bedingung ist auch bei der Anwendung der EB-Regel einzuhalten. □ steht für „widersprüchlich“ oder einfach „falsch“. Die letzte Regel besagt also, daß aus etwas falschem alles abgeleitet werden kann („ex falso quod libet“). Die problematischste der 13 Regeln ist die AB-Regel, auch bekannt als *Instantiierungsregel*. Sie erlaubt die Einführung eines beliebigen neuen Terms. In der Wahl des richtigen Terms liegt aber sehr oft die Kreativität eines Beweises, und dafür gibt dieser Kalkül keine Unterstützung.

Der Kalkül NJ ist ohne logische Axiome nicht vollständig für PL1 (für Kenner: er modelliert aber genau die intuitionistische Prädikatenlogik). Durch Hinzunahme des Satzes vom ausgeschlossenen Dritten $\mathcal{F} \vee \neg\mathcal{F}$ als einziges logisches Axiom wird aus dem Kalkül NJ der für PL1 vollständige Kalkül NK. Dies gibt ein Beispiel für die Flexibilität des Ansatzes von Gentzen, wenn es darum geht, Kalküle für neue Logiken zu entwickeln.

Beispiel: Wir leiten mit dem Gentzen-Kalkül die Formel $P\vee(Q\wedge R) \Rightarrow (P\vee Q) \wedge (P\vee R)$ ab, eines der Distributivgesetze für $\wedge$ und $\vee$:

$$\dfrac{\overset{3}{P\vee(Q\wedge R)} \qquad \dfrac{\dfrac{\overset{1}{P}}{(P\vee Q)}\,\text{OE} \qquad \dfrac{\overset{1}{P}}{(P\vee R)}\,\text{OE}}{(P\vee Q)\wedge(P\vee R)}\,\text{UE} \qquad \dfrac{\dfrac{\dfrac{\overset{2}{Q\wedge R}}{Q}\,\text{UB}}{(P\vee Q)}\,\text{OE} \qquad \dfrac{\dfrac{\overset{2}{Q\wedge R}}{R}\,\text{UB}}{(P\vee R)}\,\text{OE}}{(P\vee Q)\wedge(P\vee R)}\,\text{UE}}{\dfrac{(P\vee Q)\wedge(P\vee R)}{P\vee(Q\wedge R) \Rightarrow (P\vee Q)\wedge(P\vee R)}\,\text{FE(3)}}\,\text{OB(1,2)}$$

Die Startformeln in einem derartigen baumförmig aufgeschriebenen Beweis sind entweder logische Axiome, oder, wie im obigen Fall, beliebige Annahmen, die durch Zahlen gekennzeichnet werden. Aus der Annahme 1, also P, wird mit der OE- und UE-Regel die Formel $(P\vee Q) \wedge (P\vee R)$ abgeleitet. Dieselbe Formel ist auch aus der Annahme 2, also $Q\wedge R$, ableitbar. Die Annahme 3 ist gerade die Disjunktion der Annahmen 1 und 2, und mit der OB-Regel läßt sich jetzt die Formel $(P\vee Q) \wedge (P\vee R)$ ableiten, abhängig nur noch von der Annahme 3. Die Notation OB(1,2) drückt aus, daß man sich in diesem Schritt von den Annahmen 1 und 2 gelöst hat. Der Schritt FE(3) löst dann die letzte Abhängigkeit, so daß die letzte Formel $P \vee (Q\wedge R) \Rightarrow (P\vee Q) \wedge (P\vee R)$ von gar keinen Annahmen mehr abhängt und somit allgemeingültig ist. ■

Um die Abhängigkeit der Formeln von eventuellen Annahmen deutlicher zu machen, entwickelte Gentzen eine organisatorische Variante seines Kalküls, den sogenannten *Sequenzenkalkül*. Eine Sequenz besteht aus zwei Listen von Formeln, dem *Antezedens* $\mathcal{F}_1, \ldots, \mathcal{F}_n$ und dem *Sukzedens* $\mathcal{G}_1, \ldots, \mathcal{G}_m$, und wird mit Hilfe eines neuen Zeichens $\rightarrow$ in der Form $\mathcal{F}_1, \ldots, \mathcal{F}_n \rightarrow \mathcal{G}_1, \ldots, \mathcal{G}_m$ aufgeschrieben, mit der Bedeutung, daß unter der Annahme der Konjunktion der Antezedensformeln die Disjunktion der Sukzedensformeln abgeleitet werden kann (oder daß die

Formel $\mathcal{F}_1 \wedge \ldots \wedge \mathcal{F}_n \Rightarrow \mathcal{G}_1 \vee \ldots \vee \mathcal{G}_m$ allgemeingültig ist). Eine Sequenz enthält die vollständige Information über den momentanen Zustand einer Ableitung. Ein Beweis besteht nun aus einer Folge von Sequenzen, beginnend mit einer tautologischen Anfangssequenz der Art $\mathcal{F} \rightarrow \mathcal{F}$ und endend mit einer Sequenz $\rightarrow \mathcal{G}$, wobei $\mathcal{G}$ die zu beweisende Formel ist. Die Schlußregeln für Sequenzen sind im wesentlichen aus den Regeln für NJ abgeleitet und sollen daher hier nicht mehr alle aufgelistet werden. Die bekannteste ist die sogenannte Schnittregel:

$$\frac{\mathcal{G}_1, \ldots, \mathcal{G}_i \rightarrow \mathcal{H}_1, \ldots, \mathcal{H}_j, \mathcal{F} \qquad \mathcal{F}, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_m \rightarrow \mathcal{H}_{j+1}, \ldots, \mathcal{H}_n}{\mathcal{G}_1, \ldots, \mathcal{G}_i, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_m \rightarrow \mathcal{H}_1, \ldots, \mathcal{H}_j, \mathcal{H}_{j+1}, \ldots, \mathcal{H}_n}$$

Sie drückt in etwas allgemeinerer Form die Transitivität der Ableitungsrelation aus: Wenn aus $\mathcal{G}$ die Formel $\mathcal{F}$ ableitbar ist und aus $\mathcal{F}$ wiederum $\mathcal{H}$, dann ist aus $\mathcal{G}$ auch $\mathcal{H}$ ableitbar.

3.2 Resolution

Der Resolutionskalkül, so wie er von John Alan Robinson entwickelt wurde [Rob65a], ist ein negativer Testkalkül. Er arbeitet mit Formeln in Klauselform, besitzt als einziges logisches Axiom einen elementaren Widerspruch in Form der leeren Klausel □ und benutzt eine einzige Schlußregel, die Resolutionsregel. Die einfachste Version der Resolutionsregel hat die gleiche Gestalt wie die Schnittregel des Sequenzenkalküls im Spezialfall, daß alle Einzelformeln Literale sind (wir lassen im folgenden die Mengenklammern um Klauseln meist weg):

$$\frac{\begin{array}{ll}\text{Klausel1:} & L, K_1, \ldots, K_n \\ \text{Klausel2:} & \neg L, M_1, \ldots, M_m\end{array}}{\text{Resolvente:} \quad K_1, \ldots, K_n, M_1, \ldots, M_m}$$

Klausel1 und Klausel2 nennt man auch die *Elternklauseln* der Resolvente, L und ¬L die *Resolutionsliterale*. Die Korrektheit der Regel kann man wie folgt begründen: Gegeben sei eine Interpretation, die Klausel1 und Klausel2 erfüllt, wir haben zu zeigen, daß sie auch die Resolvente erfüllt. Eine Interpretation erfüllt eine Klausel genau dann, wenn sie wenigstens ein Literal der Klausel erfüllt (eine Klausel steht für die Disjunktion ihrer Literale). Angenommen, die gegebene Interpretation erfüllt das Literal L, dann falsifiziert sie ¬L. Da sie nach Voraussetzung wenigstens ein Literal von Klausel2 erfüllt, muß mindestens eines der Literale M_i wahr unter der Interpretation sein. Dieses Literal gehört zur Resolvente, die damit ebenfalls erfüllt wird. Wenn dagegen L falsifiziert wird, muß eines der Literale K_i und damit auch die Resolvente erfüllt sein. Also ist die Resolvente eine logische Folgerung aus den Elternklauseln.

Die volle Resolutionsregel beinhaltet zusätzlich noch eine Instantiierung der Formeln durch eine *Substitution*, das ist eine Abbildung von Variablen auf Terme. Die Anwendung der Substitution muß für die beiden Resolutionsliterale gleiche Atome ergeben. Die volle Resolutionsregel lautet:

$$\frac{\begin{array}{lll}\text{Klausel1:} & L, K_1, \ldots, K_n & \\ \text{Klausel2:} & \neg L', M_1, \ldots, M_m & \sigma L = \sigma L'\end{array}}{\text{Resolvente:} \quad \sigma K_1, \ldots, \sigma K_n, \sigma M_1, \ldots, \sigma M_m}$$

Dabei ist σ eine Substitution, die die beiden Atome L und L' gleich macht. Wenn es eine solche

Substitution für zwei Ausdrücke gibt, werden die Ausdrücke als *unifizierbar* bezeichnet und die Substitution als *Unifikator* der Ausdrücke. Zu einem Paar von unifizierbaren Ausdrücken gibt es ausgezeichnete Unifikatoren, die *allgemeinsten Unifikatoren*, aus denen sich alle anderen durch Instantiierung der verbleibenden Variablen ergeben. Eine Substitution wird als Menge von Variablenersetzungen in der Form $\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ geschrieben.

Beispielsweise sind die Terme x und f(y) unifizierbar mit allgemeinstem Unifikator $\{x \leftarrow f(y)\}$. Die Substitution $\{x \leftarrow f(a), y \leftarrow a\}$ ist auch ein Unifikator für x und f(y), aber kein allgemeinster. Er ergibt sich aus dem vorigen durch zusätzliche Instantiierung von y mit a. Zu f(x, g(x)) und f(y, g(y)) gibt es zwei äquivalente allgemeinste Unifikatoren $\{x \leftarrow y\}$ und $\{y \leftarrow x\}$, die sich nur durch Umbenennung der Variablen unterscheiden. Die Terme x und f(x) sind nicht unifizierbar (sogenannter „occur check"-Fehler), die Terme g(x) und f(x) ebensowenig („clash"-Fehler).

Robinson konnte zeigen, daß es bis auf Umbenennung der Variablen jeweils höchstens einen allgemeinsten Unifikator gibt, und daß es ausreicht, diesen bei der Anwendung der Resolutionsregel zu verwenden. Algorithmen zur Berechnung der allgemeinsten Unifikatoren werden im Abschnitt über die Unifikationstheorie vorgestellt.

Wir betrachten nun einige Beispiele für die Anwendung der Resolutionsregel.

$$\frac{\begin{array}{l} \text{ist-Mensch(Sokrates)} \\ \neg\text{ist-Mensch(x)}, \text{sterblich(x)} \qquad \sigma = \{x \leftarrow \text{Sokrates}\} \end{array}}{\text{sterblich(Sokrates)}}$$

Die zweite Elternklausel ist die Klauselform von $\forall x$ ist-Mensch(x) $\Rightarrow$ sterblich(x).

$$\frac{\begin{array}{l} P(x, a), Q(x) \\ \neg P(f(y), y), R(y) \qquad \sigma = \{x \leftarrow f(a), y \leftarrow a\} \end{array}}{Q(f(a)), R(a)}$$

$$\frac{\begin{array}{l} \text{Rasiert(x, x)}, \text{Rasiert(Barbier, x)} \\ \neg\text{Rasiert(Barbier, y)}, \neg\text{Rasiert(y, y)} \qquad \sigma = \{x \leftarrow \text{Barbier}, y \leftarrow \text{Barbier}\} \end{array}}{\text{Rasiert(Barbier, Barbier)}, \neg\text{Rasiert(Barbier, Barbier)}}$$

Dieses Beispiel ist die Klauselform der bekannten Russelschen Antinomie: Der Barbier rasiert eine Person genau dann, wenn sie sich nicht selbst rasiert. Diese Aussage ist widersprüchlich. Sie zeigt, daß noch eine Verfeinerung der Resolutionsregel notwendig ist. Der Widerspruch läßt sich nur dann ableiten, wenn die beiden Literale der Elternklauseln durch Anwenden einer Substitution gleich gemacht und verschmolzen werden, bevor die Resolvente erzeugt wird:

$$\frac{\begin{array}{lcl} \text{Rasiert(x, x)}, \text{Rasiert(Barbier, x)} & \vdash & \text{Rasiert(Barbier, Barbier)} \\ \neg\text{Rasiert(Barbier, y)}, \neg\text{Rasiert(y, y)} & \vdash & \neg\text{Rasiert(Barbier, Barbier)} \end{array}}{\square}$$

Robinson hat diese Verschmelzungsoperation ursprünglich mit in die Resolutionsregel eingebaut. Aus mehr praktischen Gründen wird sie jedoch meist als eigenständige Zusatzregel

unter dem Namen *Faktorisierung* gehandhabt.

Klassische Kalküle ergeben selbst für den aussagenlogischen Fall keine übermäßig effizienten Deduktionssysteme, da bei ihrer Entwicklung Anfang des Jahrhunderts erst einmal die Frage der prinzipiellen Übertragbarkeit von semantischem Folgern auf syntaktisches Ableiten geklärt werden mußte und an eine Implementierung noch gar nicht zu denken war. Der Resolutionskalkül stellt im Vergleich dazu einen substantiellen Fortschritt für die Anwendbarkeit von Deduktionssystemen dar. Dies liegt daran, daß er nicht mehr alle, aber noch genügend viele „interessante" Folgerungen ableitet. Sei beispielsweise die Formel $\mathcal{F} = \forall x\, P(x)$ gegeben, also gerade eine unäre Klausel. Die Resolutionsregel ist gar nicht anwendbar, es sind also keine weiteren Formeln ableitbar, obwohl unendlich viele Formeln aus $\mathcal{F}$ folgen.

Eine Folgerung aus $\mathcal{F}$ ist beispielsweise die Formel P(t) für einen beliebigen Term t. Im Gentzenkalkül kann man sie mit der AB-Regel ableiten. Mit derselben Regel sind aber auch beliebige andere Instanzen von P(x) ableitbar. Für jede Variable in einer Formel eröffnet diese Instantiierungsregel also so viele Alternativen für Ableitungen, wie es Terme gibt. Diese enorme Verzweigungsrate wird im Resolutionskalkül durch die Unifikationsidee entschärft. Variablen werden nur so weit instantiiert, wie es für die Anwendung der Resolutionsregel unbedingt nötig ist, und die Ableitungen werden auf der „allgemeinsten" Ebene durchgeführt.

Ebenso folgt aus $\mathcal{F}$ auch die Formel $\forall x\, P(x) \vee Q$. Sie läßt sich mit der OE-Regel im Gentzenkalkül ableiten. Für den Spezialfall der Klauselform erlaubt diese Regel also, aus einer Klausel eine neue abzuleiten, die beliebige weitere Literale enthält. Es ist klar, daß auch dadurch ein großer Suchraum entsteht, der im Resolutionskalkül gar nicht erst aufgespannt wird.

Aus $\mathcal{F}$ folgt auch jede Tautologie, zum Beispiel $Q \vee \neg Q$. Da man jederzeit eine elementare Tautologie als Annahme einführen kann, lassen sich im Gentzenkalkül alle Tautologien ableiten. Es ist aber gar nicht wünschenswert, sämtliche Formeln ableiten zu können, die unabhängig von $\mathcal{F}$ gelten. Aus einer unerfüllbaren Formel $\mathcal{F}$ folgt schließlich sogar jede prädikatenlogische Formel, und man ist nicht daran interessiert, diese alle aus $\mathcal{F}$ ableiten zu können. Im Resolutionskalkül entfallen auch diese Ableitungsmöglichkeiten.

Damit ist der Resolutionskalkül also nicht vollständig in dem Sinn, daß alle Folgerungen aus einer Formel auch ableitbar sind. Er besitzt jedoch die Eigenschaft der *Widerlegungsvollständigkeit* (englisch *refutation completeness*): Aus einer unerfüllbaren Klauselmenge läßt sich mit der Resolution in jedem Fall die leere Klausel, der elementare Widerspruch, ableiten. Da keine Interpretation die leere Klausel erfüllt und der Kalkül korrekt ist, gilt somit: Eine Klauselmenge ist genau dann unerfüllbar, wenn mit dem Resolutionskalkül die leere Klausel daraus ableitbar ist.

Diese Eigenschaft genügt, um alle Folgerungen nachzuweisen. Wie in Teil 2.2 gezeigt wurde, folgt aus gegebenen Formeln $\mathcal{F}_1, \ldots, \mathcal{F}_n$ eine Formel $\mathcal{B}$ genau dann, wenn $\mathcal{F}_1 \wedge \ldots \wedge \mathcal{F}_n \Rightarrow \mathcal{B}$ allgemeingültig ist. Dies wiederum ist genau dann der Fall, wenn die Formel $\mathcal{F}_1 \wedge \ldots \wedge \mathcal{F}_n \wedge \neg\mathcal{B}$ unerfüllbar ist, was äquivalent dazu ist, daß die Klauselform der letzten Formel unerfüllbar ist. Mit dem obigen Satz gilt dies genau dann, wenn aus dieser Klauselmenge die leere Klausel

ableitbar ist.

Um ein Beispiel für einen Resolutionsbeweis zu geben, formulieren wir in PL1 Aussagen über eine andere Logik, nennen wir sie MINI. Die Syntax dieser Logik MINI lasse nur nullstellige Prädikatensymbole und einen Implikationsjunktor IMP zu. Wir betrachten einen Kalkül für MINI mit zwei logischen Axiomen

$$\mathcal{X}\,\text{IMP}\,(\mathcal{Y}\,\text{IMP}\,\mathcal{X})$$
$$(\mathcal{X}\,\text{IMP}\,(\mathcal{Y}\,\text{IMP}\,\mathcal{Z}))\,\text{IMP}\,((\mathcal{X}\,\text{IMP}\,\mathcal{Y})\,\text{IMP}\,(\mathcal{X}\,\text{IMP}\,\mathcal{Z}))$$

und der Schlußregel Modus Ponens $\dfrac{\mathcal{X} \quad (\mathcal{X}\,\text{IMP}\,\mathcal{Y})}{\mathcal{Y}}$

Dabei stehen $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$ jeweils für beliebige MINI-Formeln. Wir wollen zeigen, daß im MINI-Kalkül die Formel $(\mathcal{X}\,\text{IMP}\,\mathcal{X})$ für beliebige MINI-Formeln $\mathcal{X}$ ableitbar ist.

Um die Aussagen über MINI in PL1 zu formulieren, benutzen wir ein PL1-Prädikatensymbol „ableitbar“ und kodieren den Junktor als ein PL1-Funktionssymbol „imp“ (in Präfix-schreibweise). Dadurch werden MINI-Formeln durch PL1-Terme dargestellt. Der MINI-Kalkül läßt sich dann folgendermaßen durch PL1-Formeln beschreiben:

$\mathcal{F}_1$:	$\forall$xy	ableitbar(imp(x, imp(y, x)))
$\mathcal{F}_2$:	$\forall$xyz	ableitbar(imp(imp(x, imp(y, z)), imp(imp(x, y), imp(x, z))))
$\mathcal{F}_3$:	$\forall$xy	ableitbar(x) $\wedge$ ableitbar(imp(x, y)) $\Rightarrow$ ableitbar(y)

Wir wollen zeigen, daß daraus folgende Formel folgt:

$\mathcal{B}$:	$\forall$x	ableitbar(imp(x, x)).

Die Klauselform von $\mathcal{F}_1 \wedge \mathcal{F}_2 \wedge \mathcal{F}_3 \wedge \neg\mathcal{B}$ ist eine Menge von vier Klauseln, wobei die Variable in $\neg\mathcal{B}$ durch eine nullstellige Skolemfunktion, also eine Skolemkonstante c ersetzt ist:

C1 = ableitbar(imp(x, imp(y, x)))
C2 = ableitbar(imp(imp(x, imp(y, z)), imp(imp(x, y), imp(x, z))))
C3 = ¬ableitbar(x), ¬ableitbar(imp(x, y)), ableitbar(y)
B = ¬ableitbar(imp(c,c))

Da die Variablen implizit allquantifiziert sind, können sie für jede Klausel als verschieden angenommen werden, das heißt beispielsweise, daß das x in C1 ein anderes als das x in C2 ist. Im folgenden bezeichne C,n das n-te Literal einer Klausel C, das Kürzel C,n & D,m $\vdash$ R steht für eine Anwendung der Resolutionsregel mit den entsprechenden Resolutionsliteralen der Elternklauseln C und D und der Resolvente R. Wir führen folgende Resolutionsableitung durch:

B & C3,3	$\vdash$ R1 =	¬ableitbar(x), ¬ableitbar(imp(x, imp(c,c)))
R1,2 & C3,3	$\vdash$ R2 =	¬ableitbar(x), ¬ableitbar(imp(x, imp(x', imp(c,c)))), ¬ableitbar(x')
C1 & R2,3	$\vdash$ R3 =	¬ableitbar(x),

			¬ableitbar(imp(x, imp(imp(x', imp(y',x')), imp(c,c))))
C2 & R3,2	⊢	R4 =	¬ableitbar(imp(c, (imp(imp(y, c), c)))
R4 & C1	⊢	R5 =	□

Da die leere Klausel abgeleitet werden kann, ist die Klauselmenge unerfüllbar und die Behauptung folgt wirklich aus den Voraussetzungen. Wäre □ dagegen nicht unter den ableitbaren Resolventen, wäre die Menge erfüllbar und die Behauptung würde nicht folgen. In manchen Fällen sind nur endlich viele verschiedene Resolventen aus einer Klauselmenge ableitbar, so daß man die Erfüllbarkeit tatsächlich erkennen kann. Im allgemeinen (und auch im obigen Beispiel) gibt es aber unendlich viele verschiedene ableitbare Resolventen. Durch systematische Resolventenbildung erhält man somit bestenfalls ein Semientscheidungsverfahren: Folgt die zu zeigende Behauptung wirklich, wird nach endlich vielen Resolutionsschritten die leere Klausel erreicht und die Folgerung damit nachgewiesen; folgt die Behauptung nicht, wird dies gelegentlich nach endlich vielen Schritten erkannt, im allgemeinen terminiert das Verfahren aber nicht.

Es gibt auch noch eine andere Ableitung der leeren Klausel aus der obigen Klauselmenge:

C1 & C3,1	⊢	R1 =	¬ableitbar(imp(imp(x, imp(y', x)), y)), ableitbar(y)
C2 & R1,1	⊢	R2 =	ableitbar(imp(imp(x, y'), imp(x, x)))
R2 & C3,2	⊢	R3 =	¬ableitbar(imp(x, y')), ableitbar(imp(x, x))
C1 & R3,1	⊢	R4 =	ableitbar(imp(x, x))
B & R4	⊢	R5 =	□

Das besondere an dieser Ableitung ist, daß die aus der negierten Behauptung entstandene Klausel B nur im letzten Schritt verwendet wurde und daß die vorletzte Resolvente R4 genau die Formel ist, die als Folgerung aus $\mathcal{F}_1 \wedge \mathcal{F}_2 \wedge \mathcal{F}_3$ und damit aus C1, C2, C3 nachgewiesen werden soll. In diesem Fall konnte die Behauptung also „positiv“ durch Resolution aus den Voraussetzungen abgeleitet werden.

Dies gelingt nicht immer, aber der Resolutionskalkül hat folgende wenig bekannte Eigenschaft [Kow70]: Für jede Klausel D, die aus einer Klauselmenge folgt, ist eine Klausel C ableitbar, aus der D auf eine syntaktisch einfach erkennbare Weise folgt. D kann nämlich aus C durch Instantiierung und Hinzufügen von weiteren Literalen gewonnen werden. Diese Trivialform der Folgerungsbeziehung zwischen zwei Klauseln nennt man *Subsumption*. Zum Beispiel folgt aus den Klauseln $\{P(x), Q(x)\}$ und $\{\neg P(f(y))\}$ semantisch die Klausel $D = \{Q(f(a)), R\}$. Die einzige mögliche Resolvente ist $C = \{Q(f(y))\}$. Man kann D erhalten, indem man C mit $\{y \leftarrow a\}$ instantiiert und noch das Literal R hinzufügt, das heißt, C subsumiert D. Insofern ist die Resolutionsregel noch stark genug, um genügend viele „interessante“ Folgerungen abzuleiten.

Um die Resolutionsregel auf die oben vorgestellten einfachen mehrsortigen Logiken zu übertragen, ist als einzige Änderung eine Modifikation des Unifikationsalgorithmus notwendig. Dieser muß dafür sorgen, daß für eine Variable x der Sorte S nur ein Term t eingesetzt werden kann, dessen Sorte entweder ebenfalls S oder aber eine Untersorte von S ist. Bei etwas komplizierteren Sortenbeziehungen kann das bedeuten, daß unter Umständen ein Term t zwar

nicht paßt, daß aber Instantiierungen von t mit passender Sorte gefunden werden können, indem Variable durch andere Variable mit schwächerer Sorte ersetzt werden.

Seien zum Beispiel die Sorten {Gerade, Ungerade, Nat} gegeben mit den Untersortenbeziehungen Gerade $\sqsubset$ Nat und Ungerade $\sqsubset$ Nat. Seien weiter die Argument/Ergebnissorten-Beziehungen für ein Funktionssymbol + gegeben durch

$$+: \quad (\quad \text{Gerade} \times \text{Gerade} \to \text{Gerade}, \quad \text{Ungerade} \times \text{Ungerade} \to \text{Gerade},$$
$$\text{Gerade} \times \text{Ungerade} \to \text{Ungerade}, \quad \text{Ungerade} \times \text{Gerade} \to \text{Ungerade},$$
$$\text{Nat} \times \text{Gerade} \to \text{Nat}, \quad \ldots, \quad \text{Nat} \times \text{Nat} \to \text{Nat} \quad).$$

Die Unifikation von x:Gerade mit +(y:Nat, z:Nat) ergibt zwei Lösungen, nämlich
$\{x \leftarrow +(y':\text{Gerade}, z':\text{Gerade}), \quad y \leftarrow y':\text{Gerade}, \quad z \leftarrow z':\text{Gerade}\}$ und
$\{x \leftarrow +(y':\text{Ungerade}, z':\text{Ungerade}), \quad y \leftarrow y':\text{Ungerade}, \quad z \leftarrow z':\text{Ungerade}\}$.
Diese reflektieren gerade die Tatsache, daß die Summe von zwei natürlichen Zahlen genau dann gerade ist, wenn beide Summanden gerade oder beide ungerade sind. In diesem Fall gibt es also nicht mehr einen allgemeinsten Unifikator, sondern zwei voneinander unabhängige.

Wenn es nur endlich viele Sorten gibt, findet man alle Lösungen am einfachsten, indem man systematisch für jede Variable alle Ersetzungen durch Variable mit speziellerer Sorte durchprobiert. Durch geschickte Organisation der Suche läßt sich das in der Praxis jedoch meist effizienter gestalten.

3.3 Theorieresolution

Die Theorieresolution ist ein Schema, um Information über die Bedeutung von Prädikaten- und Funktionssymbolen unmittelbar im Kalkül auszunutzen, indem man anstelle von Axiomen für diese Symbole maßgeschneiderte Schlußregeln verwendet. Sie wurde von Mark Stickel am SRI allgemein formuliert [Sti85]. Viele Spezialfälle davon waren jedoch schon vorher unter anderen Namen bekannt. Einige davon werden in den folgenden Kapiteln behandelt.

Zur Motivation der Vorgehensweise erinnern wir uns an die Begründung für die Korrektheit der einfachen Resolutionsregel:

$$\begin{array}{ll} \text{Klausel1:} & L, K_1, \ldots, K_n \\ \text{Klausel2:} & \neg L, M_1, \ldots, M_m \\ \hline \text{Resolvente:} & K_1, \ldots, K_n, M_1, \ldots, M_m \end{array}$$

Das entscheidende Argument dafür, daß die Resolvente aus den Elternklauseln folgt, war: wenn eine Interpretation das Literal L erfüllt, dann falsifiziert sie ¬L. Wesentlich ist also, daß keine Interpretation sowohl L als auch ¬L erfüllen kann. Diese Eigenschaft haben zwei Literale dann, wenn sie die rein syntaktische Bedingung erfüllen, komplementär zu sein, also übereinstimmende Prädikatensymbole und Termlisten, aber verschiedene Vorzeichen zu besitzen.

In vielen Fällen kann man den syntaktischen Komplementaritätsbegriff verallgemeinern, indem man ausnutzt, daß nicht völlig beliebige, sondern nur bestimmte Klassen von Interpretationen in Frage kommen. Zum Beispiel könnte eine Formelmenge geeignete Axiome für das

Prädikatensymbol < enthalten, so daß nur solche Interpretationen Modelle sein können, die dem Symbol < eine strikte Ordnungsrelation auf dem Universum zuordnen. Aufgrund der Eigenschaften solcher Relationen kann keine derartige Interpretation sowohl a<b als auch b<a erfüllen. In dem genannten Kontext sind diese beiden Literale zwar nicht syntaktisch, aber sozusagen semantisch komplementär, und auch folgender Ableitungsschritt ist korrekt:

Klausel1:	a<b, K
Klausel2:	b<a, M
Resolvente:	K, M

Als weitere Verallgemeinerung können wir sogar die Beschränkung auf zwei Elternklauseln aufgeben. Keine Interpretation der genannten Klasse kann sowohl a<b als auch b<c als auch c<a erfüllen. Damit kann man analog zur Begründung für die einfache Resolution, nur mit mehr Fallunterscheidungen, auch folgenden Schritt als korrekt nachweisen:

Klausel1:	a<b, K
Klausel2:	b<c, M
Klausel3:	c<a, N
Resolvente:	K, M, N

Die Idee ist also, von dem Spezialfall zweier syntaktisch komplementärer Resolutionsliterale überzugehen zu einer beliebigen Menge von Literalen, so daß keine Interpretation einer gegebenen Klasse alle diese Resolutionsliterale erfüllen kann. Die „Klasse von Interpretationen“ wird durch den Theoriebegriff präzisiert.

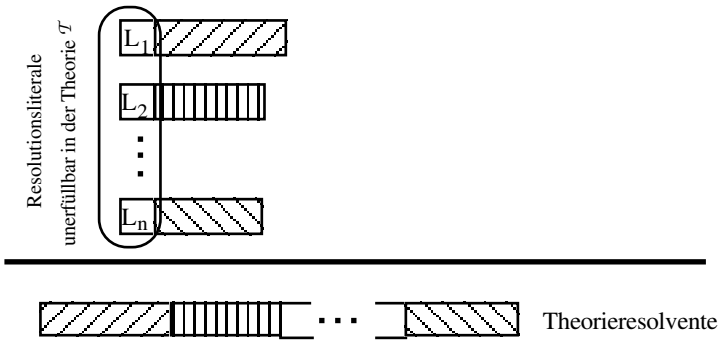
Einer erfüllbaren Menge $\mathcal{A}$ von Formeln läßt sich in PL1 eindeutig die Klasse $\mathcal{M}$ ihrer Modelle zuordnen, also von Interpretationen, die die Formeln wahr machen. Dieser Klasse von Interpretationen entspricht wiederum eindeutig eine (unendliche) maximale Menge $\mathcal{T}$ von Formeln, die von allen Interpretationen in $\mathcal{M}$ erfüllt werden. Die Formelmenge $\mathcal{T}$ ist maximal in dem Sinn, daß jede weitere Formel die Modellklasse $\mathcal{M}$ einschränkt, also von wenigstens einem Modell für $\mathcal{A}$ falsifiziert wird. Per definitionem ist $\mathcal{T}$ gerade die Menge der Folgerungen aus $\mathcal{A}$. In dieser Sichtweise enthalten $\mathcal{M}$ und $\mathcal{T}$ dieselbe Information, und beide werden oft als *Theorie* bezeichnet. Da verschiedene Formelmengen dieselben Modelle haben können, stellt ein gegebenes $\mathcal{A}$ nur eine Alternative dar, die Theorie zu definieren. Man nennt $\mathcal{A}$ auch eine *Präsentation* oder *Axiomatisierung* der Theorie.

Für eine gegebene Theorie $\mathcal{T}$ und eine Formel $\mathcal{F}$ sind die $\mathcal{T}$-*Modelle* von $\mathcal{F}$ gerade alle Modelle von $\mathcal{T}$, die auch Modelle für $\mathcal{F}$ sind. Die Begriffe $\mathcal{T}$-*Folgerung*, $\mathcal{T}$-*erfüllbar*, $\mathcal{T}$-*unerfüllbar* usw. lassen sich dann entsprechend übertragen.

Sei beispielsweise $\mathcal{A}$ die Menge $\{\forall xy\ P(x,y) \Rightarrow P(y,x)\}$, die nur aus dem Symmetrieaxiom für P besteht. Die Theorie $\mathcal{T}$ ergibt sich dann aus allen Interpretationen, die dem Prädikatensymbol P eine symmetrische Relation auf dem Universum zuordnen. Die Formel $P(a, b) \wedge \neg P(b, a)$ ist zwar erfüllbar, aber nicht $\mathcal{T}$-erfüllbar. P(b, a) ist eine $\mathcal{T}$-Folgerung von P(a, b).

Das aussagenlogische Schema für die *totale Theorieresolution* sieht nun folgendermaßen aus: Seien eine Theorie $\mathcal{T}$ und Klauseln $C_1, \ldots, C_n$ gegeben, und jede Klausel enthalte ein Literal L_i,

so daß die Konjunktion all dieser Literale $\mathcal{T}$-unerfüllbar ist. Die Vereinigung der n Klauseln abzüglich dieser Resolutionsliterale L_i ergibt eine *$\mathcal{T}$-Resolvente*. Diese ist eine $\mathcal{T}$-Folgerung von $C_1 \wedge \ldots \wedge C_n$.



Analog zur einfachen Resolution muß die Konjunktion der L_i im prädikatenlogischen Fall nicht unmittelbar $\mathcal{T}$-widersprüchlich sein. Man benötigt eine Substitution σ, einen *$\mathcal{T}$-Unifikator*, so daß die Formel $\sigma L_1 \wedge \ldots \wedge \sigma L_n$ $\mathcal{T}$-widersprüchlich ist. Die $\mathcal{T}$-Resolvente muß dann mit σ instantiiert werden. Allerdings ist ein allgemeinster $\mathcal{T}$-Unifikator für eine Menge von Ausdrücken nicht mehr bis auf Variablenumbenennung eindeutig bestimmt. Abhängig von $\mathcal{T}$ kann es einen, endlich viele oder unendlich viele voneinander unabhängige allgemeinste $\mathcal{T}$-Unifikatoren geben. Zwei Substitutionen werden dabei als unabhängig bezeichnet, wenn sich nicht eine durch Instantiierung von Variablen aus der anderen ergibt. In bösartigen Fällen existieren gar überhaupt keine allgemeinsten $\mathcal{T}$-Unifikatoren, sondern nur nicht-allgemeinste.

Eine Theorie mit endlich vielen $\mathcal{T}$-Unifikatoren ist die oben erwähnte Theorie der Symmetrie von P, die maximal zwei allgemeinste Unifikatoren erzeugt. Für

Klausel 1: P(a, b), Q
Klausel 2: ¬P(x, y), R(x)

besitzen die beiden ersten Literale die $\mathcal{T}$-Unifikatoren $\sigma_1=\{x\leftarrow a, y\leftarrow b\}$ und $\sigma_2=\{x\leftarrow b, y\leftarrow a\}$, so daß die unabhängigen $\mathcal{T}$-Resolventen {Q, R(a)} und {Q, R(b)} abgeleitet werden können.

Das Konzept der Theorieresolution erlaubt, häufig vorkommende Standardinterpretationen von Symbolen wesentlich natürlicher und effizienter als mit der üblichen Axiomatisierung und normaler Resolution zu handhaben. Die Information über die spezielle Theorie steckt dabei in erster Linie im Unifikationsalgorithmus, der allerdings für jede Theorie neu entwickelt werden muß. Um die Widerlegungsvollständigkeit der Theorieresolutionsverfahren sicherzustellen, muß der verwendete Theorieunifikationsalgorithmus *alle* allgemeinsten Unifikatoren erzeugen bzw. im unendlichen Fall zumindest aufzählen können.

Der für die Realisierung der Theorieresolution erforderliche Unifikationsalgorithmus ist für manche Theorien zu aufwendig oder gar überhaupt nicht bekannt. Dies gilt insbesondere, wenn die Theorie eigentlich aus mehreren Untertheorien besteht, die aber nicht unabhängig voneinander sind. Als Beispiel betrachten wir die Theorie $\mathcal{T}_1$ der Interpretationen, die dem Prädikatensymbol ≤ (antisymmetrische) Ordnungsrelationen zuordnen, sowie die Theorie $\mathcal{T}_2$ der Interpretationen, die dem Prädikatensymbol ≡ die Gleichheitsrelation zuordnen. Die Kombination dieser beiden Theorien macht die Konjunktion der Literale a≤b, b≤a, P(a), ¬P(b)

unerfüllbar, so daß sie ohne weiteres als Resolutionsliterale für eine Theorieresolution in Frage kommen. Ein Algorithmus, der das erkennen kann, müßte jedoch genau für die Kombination dieser beiden Theorien konzipiert sein. Sobald eine dritte hinzukommt, ist er nicht mehr anwendbar. Deshalb versucht man, Algorithmen für einzelne Theorien zu entwickeln und einen allgemeineren Mechanismus verfügbar zu machen, der den Austausch zwischen beliebigen Theorien besorgt.

Betrachten wir die Kombination der genannten Theorien und folgende Klauseln

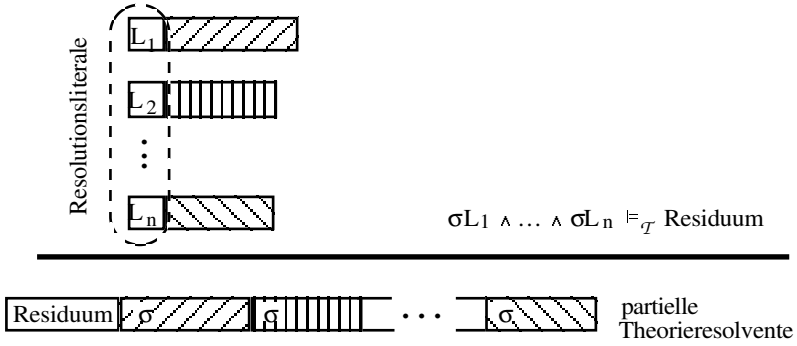
Klausel1: $a \leq b, K$
Klausel2: $b \leq a, L$
Klausel3: $P(a), M$
Klausel4: $\neg P(b), N$

Daraus müßte eigentlich die Resolvente {K, L, M, N} ableitbar sein. Dies kann man durch einen verallgemeinerten $\mathcal{T}_1$-Schritt und einen anschließenden $\mathcal{T}_2$-Schritt erreichen. Wenn eine Interpretation in der Theorie $\mathcal{T}_1$ sowohl $a \leq b$ als auch $b \leq a$ erfüllt, dann muß sie wegen der Antisymmetrie auch das Literal $a \equiv b$ erfüllen. Man sieht leicht, daß deshalb die Klausel $\{a \equiv b, K, L\}$ eine $\mathcal{T}_1$-Folgerung aus Klausel1 und Klausel2 ist. Die Literale $a \equiv b$, $P(a)$, $\neg P(b)$ können dann von dem Algorithmus für $\mathcal{T}_2$ als Resolutionsliterale für eine Gleichheitstheorieresolution zwischen der neuen Klausel sowie Klausel3 und Klausel4 erkannt werden, womit die gewünschte Klausel {K, L, M, N} entsteht.

Der erste Schritt geht über die Theorieresolution im bisherigen Sinn hinaus, da die Konjunktion der Resolutionsliterale allein noch nicht unerfüllbar in der Theorie ist, und da ein neues Literal abgeleitet und in die Resolvente aufgenommen wurde. Dieses sogenannte *Residuum* hat die Eigenschaft, daß es in der Theorie aus den Resolutionsliteralen folgt. Sein Prädikatensymbol braucht nicht einmal in den Elternklauseln vorzukommen. Ist ein Residuum beteiligt, spricht man von *partieller Theorieresolution*, andernfalls von totaler Theorieresolution.

Als Residuum kann man auch eine Disjunktion von mehreren Literalen zulassen. Das leere Residuum steht dann wie die leere Klausel für „falsch“ und folgt daher nur dann aus den Resolutionsliteralen, wenn deren Konjunktion $\mathcal{T}$-unerfüllbar ist. Dieser Spezialfall entspricht der totalen Theorieresolution.

Im allgemeinsten Fall wird die (partielle) Theorieresolution also durch folgendes Schema beschrieben:



Als Bedingung für die Widerlegungsvollständigkeit der partiellen Theorieresolution ergibt sich, daß der Unifikationsalgorithmus nicht nur alle allgemeinsten Unifikatoren, sondern auch alle „allgemeinsten Residuen“ erzeugen können muß. Genauere Untersuchungen über die Vollständigkeit bei Kombinationen von Theorien wurden bisher jedoch nicht durchgeführt.

4 Repräsentation

Die Regeln eines Kalküls geben an, wie man aus einer oder mehreren Formeln eine neue Formel ableiten kann. Im Prinzip läßt sich ein Kalkül sehr einfach in ein Computerprogramm umsetzen: Man benötigt eine Darstellung für Mengen von Formeln und Operationen für den Übergang von einer Formelmenge zu einer anderen. Wann immer eine Formelmenge $\mathcal{M}$ Formeln $\mathcal{F}_1, \ldots, \mathcal{F}_n$ enthält, aus denen mit einer Schlußregel eine Formel $\mathcal{F}$ ableitbar ist, kann ein Übergang zur Formelmenge $\mathcal{M} \cup \{\mathcal{F}\}$ erfolgen. Mit dieser *Trivialrepräsentation* entsteht also ein Suchraum, dessen Zustände Formelmengen sind.

Im Zuge einer Ableitung wachsen die Formelmengen allmählich an und enthalten daher meist auch immer mehr unnütze Teile, die den Suchraum unnötig vergrößern. Um Deduktionssysteme praktikabel zu machen, ist es deshalb dringend notwendig, neben den im Kalkül definierten Schlußregeln auch *Reduktionsregeln* zur Verfügung zu stellen, die die Formelmenge von überflüssigen Teilformeln befreien. Die gängigen Reduktionsregeln lassen sich grob in zwei Klassen einteilen:

1. Logische Vereinfachungen: Diese ersetzen Tautologien und Widersprüche durch die Wahrheitswerte „wahr“ und „falsch“ und andere Formeln durch strukturell einfachere, aber logisch äquivalente Formeln. Beispielsweise kann die Formel $\forall x\, P(x) \vee P(a)$ durch die äquivalente Formel P(a) ersetzt werden, während die Formel $\forall x\, P(x) \wedge P(a)$ zu $\forall x\, P(x)$ vereinfacht werden kann.

2. Elimination „nutzloser“ Formeln: Ein typischer Vertreter hiervon ist die *Isolationsregel* (englisch *purity principle*) im Resolutionskalkül: Enthält eine Klausel ein Literal, das mit keinem anderen Literal in der Klauselmenge resolvierbar ist, so sind daraus nur Resolventen ableitbar, die wieder ein solches isoliertes Literal enthalten. Die Klausel kann daher nicht zur Ableitung der leeren Klausel beitragen und darf als „nutzlos“ gelöscht werden.

Beim Entwickeln solcher Reduktionsregeln sind der Findigkeit des Konstrukteurs eines Deduktionssystems kaum Grenzen gesetzt. Ob sie im konkreten Einzelfall auch tatsächlich angewandt werden dürfen, hängt jedoch im allgemeinen nicht nur vom logischen Status einer Formel ab, sondern auch vom momentanen Gesamtzustand des Suchverfahrens, das unter Umständen mit der unreduzierten Formelmenge erfolgreich weiterarbeiten kann, mit der vereinfachten Version jedoch scheitert. Bei der Vorstellung des Klauselgraphverfahrens werden wir auf solche Schwierigkeiten eingehen.

4.1 Tableaus

Das erste Verfahren, bei dem über Formelmengen hinaus komplexere Strukturen zur Repräsentation des Suchzustandes eingesetzt werden, ist das Tableauverfahren für Gentzen-Kalküle. Gentzen-Kalküle sind positive deduktive Kalküle, d.h. ausgehend von elementaren logischen Tautologien werden immer komplexere Formeln hergeleitet, die per Konstruktion ebenfalls Tautologien sind. Man kann daraus einen Testkalkül konstruieren, indem man die Gentzenregeln einfach umdreht und die Behauptung soweit in ihre Teilformeln zerlegt, bis sich über deren logischen Status eine Aussage machen läßt. Nicht ganz unproblematisch sind Formeln wie $\mathcal{F} \vee \mathcal{G}$, die Alternativen und damit Indeterminismen ermöglichen. Sie erzwingen, die Zerlegung baumförmig zu organisieren, so daß in jedem Zweig der jeweilige Fall einzeln verfolgt werden kann.

Nach einigen Ansätzen von Jaakko Hintikka und Evert Beth hat schließlich Raymond Smullyan mit seinen *analytischen Tableaus* diese Idee zu einem negativen Testkalkül ausgearbeitet [Smu68], der heute weit verbreitet ist. Das Verfahren baut ausgehend von der Negation der als allgemeingültig nachzuweisenden Formel durch Anwendung von Dekompositionsregeln auf die Formeln einen Baum auf. Die Dekomposition endet, wenn die Formel in ihre atomaren Bestandteile zerlegt ist und entweder jeder einzelne Zweig als widersprüchlich erkannt wurde, oder zumindest ein Zweig eine konsistente Belegung der atomaren Formeln mit Wahrheitswerten zuläßt, aus dem sich dann ein Modell ergibt. Im ersten Fall ist die negierte Formel widersprüchlich, und damit die unnegierte Formel allgemeingültig; im zweiten Fall hat man ein Gegenbeispiel gefunden. Bei einer erfüllbaren Wurzel terminiert die Zerlegung nicht notwendigerweise, sonst ergäbe sich ein Entscheidungsverfahren für PL1. Die Dekompositionsregeln garantieren, daß alle Fallunterscheidungen systematisch untersucht werden und kein Fall vergessen werden kann.

Die Dekompositionsregeln sind:

A:	B:	C:	D:	E:	F:	G:
$\neg\neg\mathcal{F}$	$\mathcal{F}\wedge\mathcal{G}$	$\neg(\mathcal{F}\vee\mathcal{G})$	$\neg(\mathcal{F}\Rightarrow\mathcal{G})$	$\mathcal{F}\vee\mathcal{G}$	$(\mathcal{F}\Rightarrow\mathcal{G})$	$\neg(\mathcal{F}\wedge\mathcal{G})$
$\mathcal{F}$	$\mathcal{F}$ $\mathcal{G}$	$\neg\mathcal{F}$ $\neg\mathcal{G}$	$\mathcal{F}$ $\neg\mathcal{G}$	$\mathcal{F} \mid \mathcal{G}$	$\neg\mathcal{F} \mid \mathcal{G}$	$\neg\mathcal{F} \mid \neg\mathcal{G}$

H:	I:	J:	K:	L:	M:
$\mathcal{F}\Leftrightarrow\mathcal{G}$	$\neg(\mathcal{F}\Leftrightarrow\mathcal{G})$	$\forall x\ \mathcal{F}(x)$	$\neg\exists x\ \mathcal{F}(x)$	$\exists x\ \mathcal{F}(x)$	$\neg\forall x\ \mathcal{F}(x)$
$\mathcal{F} \mid \neg\mathcal{F}$ $\mathcal{G} \mid \neg\mathcal{G}$	$\mathcal{F} \mid \neg\mathcal{F}$ $\neg\mathcal{G} \mid \mathcal{G}$	$\mathcal{F}(t)$	$\neg\mathcal{F}(t)$	$\mathcal{F}(a)$	$\neg\mathcal{F}(a)$

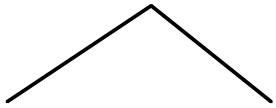
Bei den Instantiierungsregeln J und K darf der Term t keine Variable enthalten, die in $\mathcal{F}$ gebunden ist, bei L und M muß a eine neue Konstante sein. Die Diagramme sind folgendermaßen zu lesen: Um die Formel über dem Strich zu widerlegen, widerlege die unter dem Strich stehenden. Stehen zwei durch einen senkrechten Strich getrennte Formeln nebeneinander, müssen beide widerlegt werden. Von untereinanderstehenden Formeln reicht die Widerlegung

einer einzigen. Als Anweisung zur Konstruktion des Tableaus lesen sie sich dementsprechend: Wenn im gerade bearbeiteten Zweig des Tableaus eine Formel vorkommt, die einem Typ entspricht, wie er über dem Strich steht, so erweitere den Zweig, indem die unter dem Strich untereinanderstehenden Formeln als neue Blätter angehängt werden und die nebeinanderstehenden Formeln zwei neue Zweige eröffnen. Ein Zweig heißt *beendet*, wenn keine Regel mehr anwendbar ist (*offen*), oder wenn in ihm eine Formel und ihre Negation vorkommen (*geschlossen*). Das Verfahren terminiert, wenn alle Zweige beendet sind.

Beispiel: Zu zeigen sei, daß, wenn Politiker niemanden mögen, der knausrig ist und jeder Politiker eine Firma mag und es überhaupt Politiker gibt, dann auch eine Firma existiert, die nicht knausrig ist. Die diesen Aussagen entsprechende Formel ist also als allgemeingültig nachzuweisen. Ihre Negation bildet somit die Wurzel des Tableaus. Die Struktur der ersten vier Zeilen ergibt sich mit den Regeln D und B aus jeder Formel des Typs ¬[Voraussetzungen ⇒ Behauptung]. Die mit *n gekennzeichneten Blätter des Baumes zeigen an, mit welcher Zeile der Zweig jeweils geschlossen wurde.

¬[(∀x Politiker(x) ⇒ (∀y Knausrig(y) ⇒ ¬Mag(x,y)))
∧ (∀x Politiker(x) ⇒ ∃y Firma(y) ∧ Mag(x,y))
∧ (∃x Politiker(x))
⇒ (∃x Firma(x) ∧ ¬Knausrig(x))] negierte Formel

1)	∀x Politiker(x) ⇒ (∀y Knausrig(y) ⇒ ¬Mag(x,y))		
2)	∀x Politiker(x) ⇒ ∃y Firma(y) ∧ Mag(x,y)		
3)	∃x Politiker(x)		
4)	¬∃x (Firma(x) ∧ ¬Knausrig(x))		Regeln B, D
5)	Politiker(WG)		Regel L auf 3
6)	Politiker(WG) ⇒ ∃y Firma(y) ∧ Mag(WG,y)		Regel J auf 2
7)	¬Politiker(WG)	∃y Firma(y) ∧ Mag(WG,y)	Regel F auf 6
8)	*5	Firma(F) ∧ Mag(WG,F)	Regel L auf 7
9)		Firma(F)	
10)		Mag(WG,F)	Regel B auf 8
11)	¬(Firma(F) ∧ ¬Knausrig(F))		Regel K auf 4
12)	¬Firma(F) *9	Knausrig(F)	Regeln G, A auf 11
13)	Politiker(WG) ⇒ (∀y Knausrig(y)⇒¬Mag(WG,y))		Regel J auf 1
14)	¬Politiker(WG)	∀y Knausrig(y) ⇒ ¬Mag(WG,y)	Regel F auf 13
15)	*5	Knausrig(F) ⇒ ¬Mag(WG,F)	Regel J auf 14



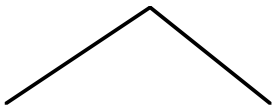
16)		¬Knausrig(F)	¬Mag(WG,F)	Regel F auf 15
		*12	*10	

Da alle Zweige geschlossen sind, ist die negierte Formel an der Wurzel unerfüllbar, die unnegierte also allgemeingültig. ■

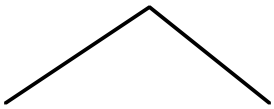
Das nächste Beispiel zeigt, wie sich bei einer erfüllbaren Formel ein Modell konstruieren läßt.

Beispiel: Zu zeigen sei: wenn es jemanden gibt, der umweltbewußt und intelligent ist und wenn es einen intelligenten Umweltminister gibt, dann gibt es auch einen umweltbewußten Umweltminister.

	¬[	(∃x Umweltbewußt(x) ∧ Intelligent(x))		
	∧	(∃x Umweltminister(x) ∧ Intelligent(x))		
	⇒	(∃x Umweltbewußt(x) ∧ Umweltminister(x))]		negierte Formel
1)		∃x Umweltbewußt(x) ∧ Intelligent(x)		
2)		∃x Umweltminister(x) ∧ Intelligent(x)		
3)		¬ ∃x Umweltbewußt(x) ∧ Umweltminister(x)		Regeln B, D
4)		Umweltbewußt(F) ∧ Intelligent(F)		Regel L auf 1
5)		Umweltbewußt(F)		
6)		Intelligent(F)		Regel B auf 4
7)		Umweltminister(W) ∧ Intelligent(W)		Regel L auf 2
8)		Umweltminister(W)		
9)		Intelligent(W)		Regel B auf 7
10)		¬(Umweltbewußt(F) ∧ Umweltminister(F))		Regel K auf 3



11)	¬Umweltbewußt(F)		¬Umweltminister(F)	Regel G auf 10
12)	*5		¬(Umweltbewußt(W) ∧ Umweltminister(W))	Regel K auf 3



13)		¬Umweltbewußt(W)	¬Umweltminister(W)	Regel G auf 12
		offen	*8	

Mit der Instantiierungsregel K sind alle Terme, die im offenen Zweig auftreten, eingeführt, daher kann man die Regelanwendung abbrechen und erhält aus diesem Zweig „W“ als Gegenbeispiel, für das Umweltminister(W) und ¬Umweltbewußt(W) gilt. Damit folgt die Behauptung nicht notwendigerweise aus den Voraussetzungen. Das schließt jedoch nicht aus, daß es Interpretationen gibt, unter denen die Behauptung wahr ist. ■

Das ursprünglich von Smullyan entwickelte Tableauverfahren enthält, besonders durch die unspezifische Instantiierungsregel, so viele Verzweigungsmöglichkeiten, daß es bisher wenig praktischen Erfolg hatte. Durch die leichte Modifizierbarkeit der Dekompositionsregeln ist es jedoch so flexibel, daß es leicht an andere Logiken angepaßt werden kann. Raymond Smullyan

und Melvin Fitting haben das Verfahren für verschiedene Modal- und intuitionistische Logiken formuliert [Fit83]. Ein auf Tableaus basierender automatischer Beweiser mit starken Heuristiken zur Auswahl der Dekompositionsregeln und zum Erkennen von Sackgassen wurde von F. Oppacher und E. Suen an der Carleton University in Ottawa entwickelt [OS86]. Viele der Indeterminismen beim Tableauverfahren wurden inzwischen ausgemerzt. Z.B. wurde Unifikation eingeführt [Wri 89b, Fit 90] und neue Abarbeitungsstrategien entwickelt [Wri 89a]. Ein Beweiser, der viele dieser Ideen integriert hat, ist der an der TU München entwickelte SETHEO Beweiser [LeSc92]. In der konkreten technischen Realisierung ist in solchen Systemen dann kaum noch auszumachen, ob es sich um einen Tableaubeweiser, einen Konnektionsbeweiser oder einen Modell Eliminationsbeweiser handelt.

4.2 Klauselgraphen

Wir wollen uns nun einer Repräsentationsform für den Resolutionskalkül zuwenden, die ursprünglich von Robert Kowalski entwickelt wurde [Kow75]. Sein „connection graph"-Verfahren ist nur für die einfache Resolution definiert. Wir werden es jedoch in einer etwas erweiterten Form vorstellen, und zwar für die eingeschränkte Klasse der totalen Theorieresolution unter Theorien, für die immer endliche Mengen von allgemeinsten Unifikatoren existieren.

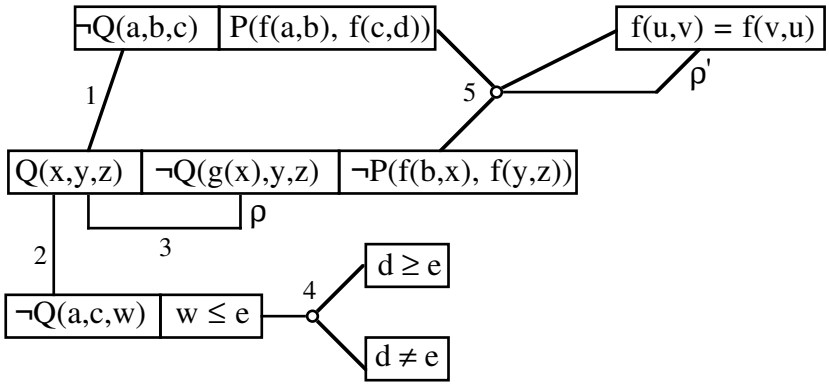
4.2.1 Klauselgraphen als Datenstruktur und Indexierungshilfsmittel

Ein Klauselgraph basiert auf einer Menge von Knoten, die mit Literalen markiert sind. Diese *Literalknoten* werden zusammengruppiert zu *Klauselknoten*, die Mengen von Literalen, also Klauseln, repräsentieren. Üblicherweise stellt man die Literalknoten graphisch durch kleine Kästchen dar, in die die markierenden Literale hineingeschrieben werden, und die Klauselknoten durch mehrere zusammenhängende solche Kästchen. Beliebige Relationen zwischen Literalen können nun durch Kanten zwischen Literalknoten repräsentiert werden. Für die wichtigste Relation, die Resolvierbarkeitsrelation, stehen sogenannte *R-Kanten*. Diese verbinden die Resolutionsliterale, die an einem (Theorie-)Resolutionsschritt beteiligt sein können, und werden meist selbst mit den allgemeinsten Unifikatoren markiert.

Die Unterscheidung zwischen Literalknoten und Literalen sowie Klauselknoten und Klauseln hat mehr technische Gründe. Verschiedene Knoten können durchaus mit demselben Literal markiert, aber mit völlig unterschiedlichen Kanten verbunden sein. Würde man die Literale selbst als Knoten des Graphen auffassen, ließen sich derartige Phänomene überhaupt nicht formulieren. Soweit Verwechslungen ausgeschlossen sind, werden wir im folgenden aber nicht streng zwischen Knoten und Formeln unterscheiden.

Beispiel: Der nachfolgende Klauselgraph enthält sechs Klauselknoten. Die R-Kante 1 verbindet zwei Resolutionsliterale für einen einfachen Resolutionsschritt mit dem Unifikator $\{x \leftarrow a, y \leftarrow b, z \leftarrow c\}$. Bei Abarbeitung dieses Schritts entstünde die Resolvente $\{P(f(a,b), f(c,d)), \neg Q(g(a),b,c), \neg P(f(b,a),f(b,c)\}$. Die R-Kante 2 verbindet ebenfalls zwei einfache

Resolutionsliterale mit zugehörigem Unifikator {x←a, y←c, z←w}.



Die R-Kante 3 verbindet zwei Literale innerhalb derselben Klausel, die zwar gleiches Prädikat und verschiedenes Vorzeichen haben, aber zunächst nicht unifizierbar sind. Diese R-Kante zeigt eine Resolutionsmöglichkeit mit einer Kopie der Klausel an, in der die Variablen durch eine Substitution ρ = {x←x', y←y', z←z'} umbenannt sind, also {Q(x',y',z'), ¬Q(g(x'),y',z'), ¬P(f(b,x'),f(y',z'))}. Das erste Literal des Originals ist nun mit dem zweiten Literal der Kopie resolvierbar mit dem Unifikator {x←g(x'), y←y', z←z'}, wodurch {Q(x',y',z'), ¬P(f(b,x'),f(y',z')), ¬Q(g(g(x')),y',z'), ¬P(f(b,g(x')),f(y',z')} als Resolvente entstünde. Der analoge Schritt zwischen dem ersten Literal der Kopie und dem zweiten Literal des Originals führt zu einer Resolvente, in der die gestrichenen und ungestrichenen Variablen gegenüber der vorigen gerade vertauscht sind, daher genügt eine der beiden Varianten. Eine derartige *Selbstresolution* einer Klausel mit einer Kopie ihrer selbst wird in Implementierungen praktisch immer unterlassen. Im Fall der Theorieresolution ist sie allerdings notwendig.

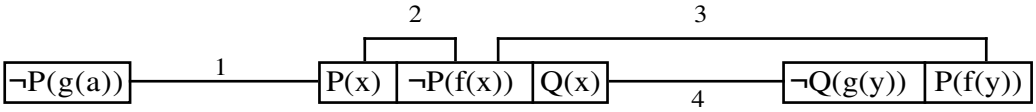
Die R-Kante 4 ist eine echte Theorie-R-Kante. In der Theorie der Ordnungsrelationen und der Gleichheit wird die Konjunktion der Literale w≤e, d≥e, d≠e widersprüchlich, wenn man sie mit der Substitution {w←d} instantiiert. Damit ist die Theorieresolvente {¬Q(a,c,d)} ableitbar. Die R-Kante 5 schließlich involviert zwei verschiedene Varianten der Kommutativitätsklausel, {f(u,v) = f(v,u)} und {f(u',v') = f(v',u')}, und ist mit zwei allgemeinsten Unifikatoren markiert: {x←a, y←c, z←d, u←a, v←b} und {x←a, y←d, z←c, u←a, v←b, u'←c, v'←d}. Der erste Unifikator entspricht einer Anwendung des Kommutativitätsgesetzes auf den ersten Unterterm f(a,b), um ihn mit f(b,x) zu unifizieren, während f(c,d) und f(y,z) ohne Vertauschungen unifiziert werden. Der zweite Unifikator benutzt die Kommutativität ein zweites Mal, um auch die Argumente von f(c,d) zu vertauschen. ■

Durch die Klauselgraphrepräsentation ergibt sich zunächst ein erster einsichtiger Vorteil. In den traditionellen Verfahren sind die Resolutionsreihenfolgen relativ fest durch den Suchalgorithmus vorgegeben. Die explizite Repräsentation der Resolutionsmöglichkeiten als R-Kanten im Klauselgraphen ermöglicht es dagegen, vor der Ausführung eines Resolutionsschrittes alle R-Kanten zu bewerten und nach heuristischen Kriterien die günstigste Wahl zu treffen.

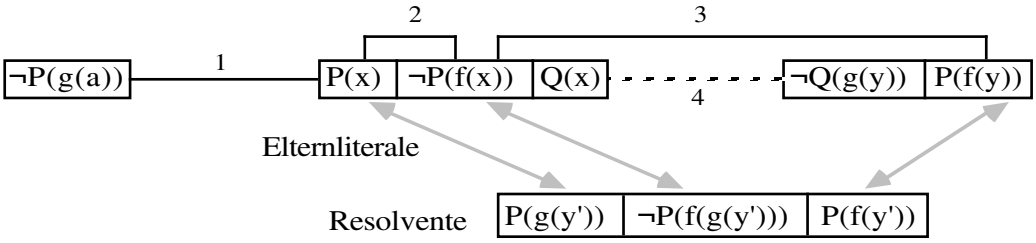
Eine naive Übertragung der Resolutionsoperation auf Klauselgraphen ist folgendermaßen möglich: Man erzeuge die durch eine R-Kante angezeigte Resolvente in der üblichen Weise, bilde den dazugehörigen Klauselknoten, wobei die Variablen der Resolvente gegebenenfalls

durch neue ersetzt werden, und berechne die neuen R-Kanten, indem man alle Resolutionsmöglichkeiten der neuen mit den schon vorhandenen Literalen untersucht. Der letzte Schritt ist sehr aufwendig und läßt sich durch eine einfache Überlegung ganz wesentlich verbessern: Die Literale in der Resolvente sind Instanzen von Literalen in den Elternklausel, den sogenannten *Elternliteralen*. Daher kann es keine Resolutionsmöglichkeiten mit den neuen Literalen geben, wo es nicht schon entsprechende Resolutionsmöglichkeiten mit den Elternliteralen gab. Es reicht also aus, für die Resolvente alle mit den R-Kanten der Elternliterale verbundenen Literale auf neue Resolutionsmöglichkeiten zu untersuchen, und so die neuen Kanten aus den alten zu *vererben*.

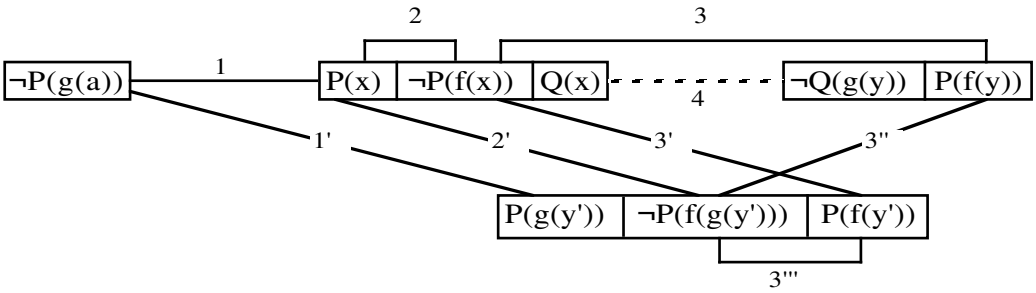
Beispiel: Vererbung von R-Kanten



In diesem Ausgangsgraphen wird „auf“ R-Kante 4 resolviert, wodurch zunächst folgende Situation entsteht:



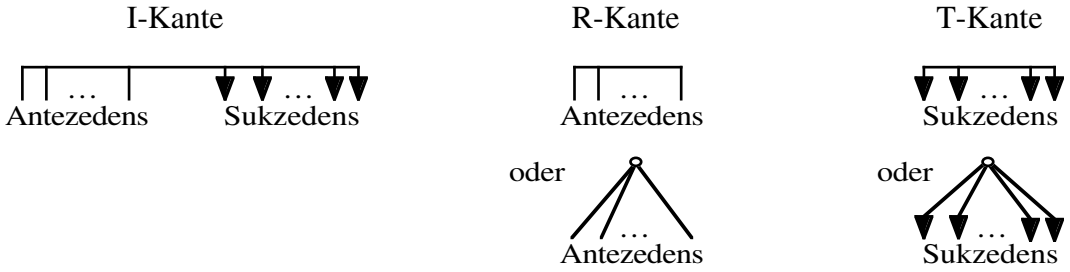
Der neue Klauselgraph ergibt sich durch Vererbung der alten Kanten 1, 2, 3:



Der zweite Vorteil der Klauselgraphrepräsentation liegt also darin, daß die R-Kanten einen vorzüglichen Indexierungsmechanismus bieten, um für eine Resolvente die Resolutionsmöglichkeiten mit den alten Klauseln zu berechnen. Für eine relativ große Klasse von Theorien ist es sogar möglich, die Unifikatoren der neuen R-Kanten direkt aus den Unifikatoren der alten Kanten zu berechnen, ohne die Literale neu unifizieren zu müssen (vgl. [Ohl87]).

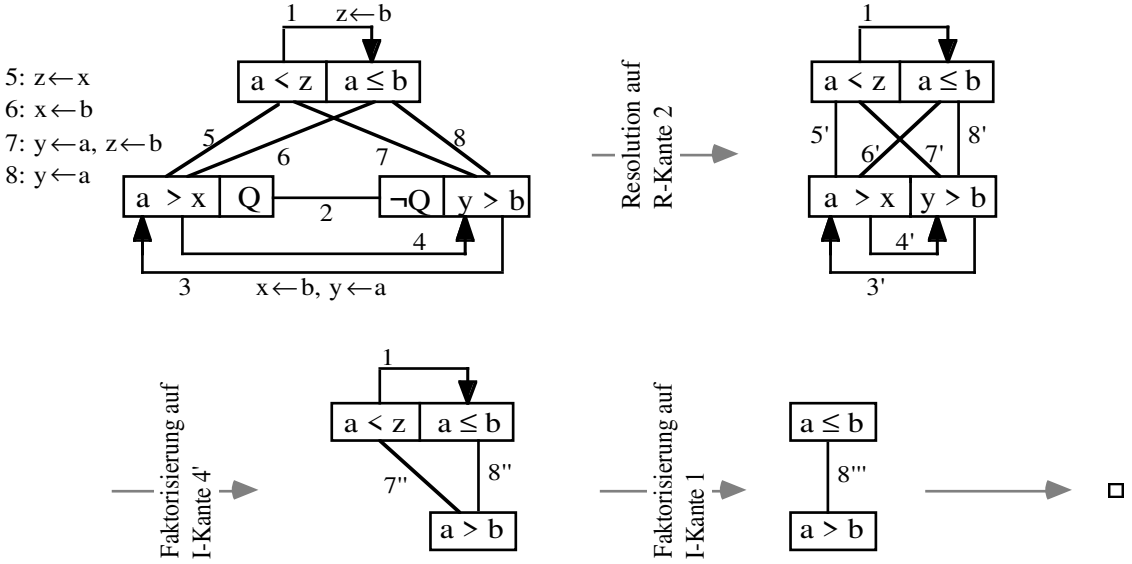
Die bisherigen R-Kanten sind nur ein Sonderfall einer allgemeineren Form von Kanten. Eine kurze Überlegung zur logischen Bedeutung der R-Kante mag dies verdeutlichen. Wenn eine R-Kante für die Theorie $\mathcal{T}$ die Literale $L_1, \ldots, L_n$ verbindet und mit einem Unifikator σ markiert ist, bedeutet das, daß die Konjunktion der Literale $\sigma L_1, \ldots, \sigma L_n$ $\mathcal{T}$-widersprüchlich ist. Dies ist genau dann der Fall, wenn die Formel $\sigma L_1 \wedge \ldots \wedge \sigma L_n \Rightarrow \square$ in $\mathcal{T}$ allgemeingültig ist. Eine

naheliegende Verallgemeinerung besteht darin, zwei Gruppen von Literalen, das *Antezedens* $L_1, \ldots, L_n$ und das *Sukzedens* $K_1, \ldots, K_m$ durch eine sogenannte Implikations- oder einfach *I-Kante* zu verbinden, wenn die Formel $\sigma L_1 \wedge \ldots \wedge \sigma L_n \Rightarrow \sigma K_1 \vee \ldots \vee \sigma K_m$ in $\mathcal{T}$ allgemeingültig ist. Bei leerem Sukzedens liegt gerade der obige Spezialfall der R-Kante vor, die Resolutionsmöglichkeiten anzeigt. Der andere Spezialfall mit leerem Antezedens besagt dann, daß $\sigma K_1 \vee \ldots \vee \sigma K_m$ in $\mathcal{T}$ allgemeingültig ist, und zeigt damit Tautologien an. Derartige Kanten heißen daher *T-Kanten*. Graphisch stellen wir die Kanten folgendermaßen dar:



Der allgemeine Fall einer I-Kante entspricht einer partiellen Theorieresolution, bei der das Residuum die Instanz $\sigma K_1 \vee \ldots \vee \sigma K_m$ des Sukzedens ist. Diese Deutung der Kantentypen erfordert, daß die Antezedensliterale in verschiedenen Klauseln (die ja konjunktiv verknüpft sind) und die Sukzedensliterale innerhalb einer Klausel (die ja eine Disjunktion ist) liegen. Mehrere Antezedensliterale in einer Klausel werden so aufgefaßt, daß sie zu verschiedenen Kopien derselben Klausel gehören. Sind Sukzedensliterale über verschiedene Klauseln verteilt, darf die I-Kante nicht abgearbeitet werden. Andere Schritte können aber Instanzen dieser Sukzedenzliterale in dieselbe Resolvente wandern lassen, so daß durch Vererbung eine abarbeitbare I-Kante entsteht. Wenn Antezedens- und Sukzedensliterale in derselben Klausel liegen, kann man daraus eine Klausel ableiten, indem man die Antezedensliterale entfernt und den Rest entsprechend instantiiert. Diese Operation entspricht einer gerichteten Faktorisierung.

Beispiel: Abarbeitung und Vererbung von I-Kanten



Die I-Kante 1 ist eine echte Theorie-I-Kante für die Theorie der Ordnungsrelationen und repräsentiert die Implikation $a<b \Rightarrow a \leq b$. Die beiden I-Kanten 3 und 4 spiegeln dagegen einfach

die aussagenlogische Äquivalenz $a>b \Leftrightarrow a>b$ wider und enthalten damit eine gewisse Redundanz. Die Darstellung der Äquivalenz durch zwei Implikationen gibt jedoch mehr Freiheit in der Ausführung der Faktorisierung, wie wir im zweiten Schritt sehen werden. Zunächst wird auf R-Kante 2 resolviert. Die Resolvente und die vererbten Kanten sind im zweiten Graphen dargestellt, aus Platzgründen lassen wir die Elternklauseln jeweils weg. Die beiden neuen I-Kanten 3' und 4' entstehen durch Vererbung aus den I-Kanten 3 und 4. Im zweiten Schritt wird mit I-Kante 4' der Faktor $a>b$ abgeleitet, der im Prinzip auch aus I-Kante 3' erzeugt werden könnte. Da bei der Bildung des Faktors einfach die Antezedensliterale weggestrichen werden, würden sich jedoch unterschiedliche Graphen ergeben, wenn die beiden Elternliterale unterschiedliche Kanten besäßen, und dies ist wegen der im nächsten Abschnitt diskutierten Reduktionsregeln möglich. Die in den letzten beiden Schritten dargestellten Operationen sind selbsterklärend. ■

Insgesamt ergibt sich damit ein Verfahren zum Nachweis der Unerfüllbarkeit einer Formel $\mathcal{F}$ in Klauselform, die sogenannte *Klauselgraphresolution*. Zunächst wird der initiale Klauselgraph für die Klauselmenge aufgebaut, indem alle zwischen den Literalknoten möglichen Kanten berechnet werden. Es folgt eine Abarbeitungsphase, in der Operationen auf diesen Kanten ausgeführt werden. Eine Operation erzeugt eine neue Klausel, etwa eine Resolvente oder einen Faktor, sowie durch Vererbung neuen Kanten. In dem ursprünglich von R. Kowalski definierten Verfahren wird außerdem die abgearbeitete Kante jeweils entfernt, um eine Wiederholung desselben Schritts zu vermeiden. Das Verfahren endet, wenn entweder ein Graph erreicht ist, der die leere Klausel enthält, oder ein völlig kantenloser Graph, auf den also keine Operation anwendbar ist. Im ersten Fall ist $\mathcal{F}$ unerfüllbar, im zweiten erfüllbar. Werden die im folgenden beschriebenen Reduktionsregeln mit einbezogen, sind nur in einem einzigen Fall keine Operationen anwendbar, nämlich für den leeren Klauselgraphen, der weder Kanten noch Klauseln (nicht einmal die leere Klausel) enthält.

4.2.2 Klauselgraphen mit allgemeinen Reduktionsregeln

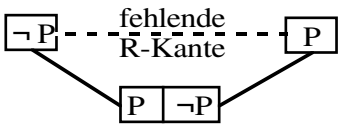
Die Datenstrukturen eines Klauselgraphen eignen sich sehr gut, um die zu Beginn des Abschnitts 4 erwähnten Reduktionsregeln effizient zu realisieren. Wir wollen die wichtigsten der Reduktionsregeln, die für Resolutionskalküle allgemein anwendbar sind, nämlich Tautologielöschung, Subsumption und Literallöschungen, in der Klauselgraphversion vorstellen.

Eine *Tautologieklausel* ist eine (in der jeweiligen Theorie) allgemeingültige Disjunktion. Sie wird durch T-Kanten mit „leerem“ Unifikator, also der identischen Substitution, angezeigt.

Allgemeines Schema für Tautologieerkennung: Beispiele:

Tautologische Klauseln sind aus logischer Sicht nutzlos bei der Suche nach einem Widerspruch und sollten daher aus der jeweiligen Formelmenge entfernt werden dürfen. Im Rahmen eines

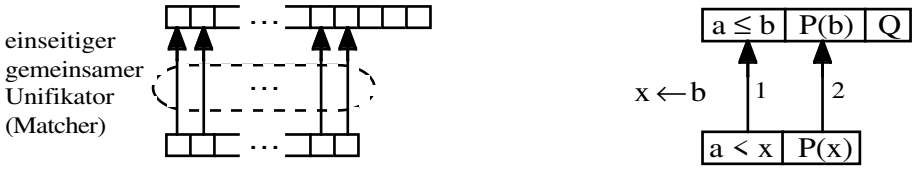
komplexeren Suchverfahrens ist jedoch nicht nur der logische Status einer Formel wichtig, sondern auch ihr Umfeld innerhalb des Ableitungsprozesses. Für das Klauselgraphverfahren wird das Umfeld durch das Vorhandensein bzw. Nichtvorhandensein einer Kante bestimmt. Kantenlöschregeln, wie sie im nächsten Abschnitt beschrieben werden, können dazu führen, daß resolvierbare Literale nicht mehr mit einer Kante verbunden sind. Als Folge davon kann es beispielsweise passieren, daß Tautologien trotzdem zur Herleitung der leeren Klausel notwendig werden, wie das folgende Beispiel zeigt:



Durch Resolution auf der ersten R-Kante und anschließende Resolution auf dem Nachfolger der zweiten R-Kante ergibt sich die leere Klausel. Würde die Tautologieklausel entfernt, entstünde ein Klauselgraph mit zwei komplementären unären Klauseln, aber keinen Kanten. Die leere Klausel wäre also nicht mehr ableitbar. Um sicherzustellen, daß auf die Tautologie verzichtet werden kann, muß deshalb folgende Bedingung eingehalten werden: Wenn die Literale $L_1 \vee \ldots \vee L_n$ eine Tautologie in der Theorie $\mathcal{T}$ darstellen, dann sind die von der Tautologie aus über normale R-Kanten (zur leeren Theorie) erreichbaren Literale $\neg L_1 \wedge \ldots \wedge \neg L_n$ $\mathcal{T}$-widersprüchlich und müßten ihrerseits mit einer R-Kante verbunden sein. Falls das nicht der Fall ist, darf entweder die Tautologie nicht entfernt werden, oder die fehlenden Kanten müssen wieder eingefügt werden.

Die zweite wichtige Reduktionsregel, die *Subsumption*, ist ein syntaktisch leicht erkennbarer Spezialfall der Implikation. Die ursprüngliche Definition (ohne Theorien) ist: Eine Klausel C subsumiert eine Klausel D, falls eine Substitution σ existiert, so daß $\sigma C \subseteq D$ gilt. Zum Beispiel subsumiert {P(x,x)} die Klausel {P(a,a), Q}, aber nicht die Klausel {P(u,v), Q}. Unter Theorien läßt sich die Definition etwas erweitern: Beim Test $\sigma C \subseteq D$ werden die Literale nicht mehr nur auf syntaktische Gleichheit, sondern auf Implikation in der Theorie überprüft. Im Klauselgraphen werden solche Implikationen durch I-Kanten angezeigt.

Allgemeines Schema für Subsumptionserkennung: Beispiel:



Dabei subsumiert die untere Klausel jeweils die obere. Die subsumierte Klausel, also die längere, kann normalerweise entfernt werden. Allerdings werden auch Faktoren stets von ihren Elternklauseln subsumiert, ohne deswegen überflüssig zu sein. Bei der Anwendung der Subsumption ist also wieder das Umfeld der Klausel im Ableitungsprozeß zu beachten. Für die Klauselgraphresolution kommt eine Bedingung an die Kanten hinzu: eine subsumierte Klausel darf nur dann entfernt werden, wenn alle R-Kanten dieser Klausel ein entsprechendes Pendant in der subsumierenden Klausel haben.

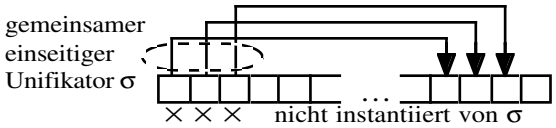
Die dritte Klasse von Reduktionsregeln modifiziert einzelne Klauseln durch *Literallöschungen*.

Literale können immer dann aus einer Klausel gelöscht werden, wenn die kürzere Klausel zur längeren Klausel logisch äquivalent ist. Im Gegensatz zum Entfernen ganzer Klauseln erfordern Literallöschungen keine Rücksicht auf existierende oder nichtexistierende Kanten eines Klauselgraphen.

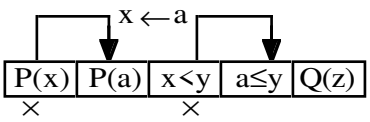
Besonders einfach ist die *Literalverschmelzung*: Von zwei syntaktisch übereinstimmenden Literalen in einer Klausel kann eins wegfallen. Diese Anwendung des Idempotenzgesetzes für die Disjunktion ist zwar durch die formale Definition einer Klausel als Menge von Literalen sozusagen automatisch eingebaut, sie muß aber in einem realen Programm in jedem Fall implementiert werden. Die eingangs beschriebene Sichtweise eines Klauselknoten als Menge von Literalknoten, die mit gleichen Literalen markiert sein können, legt ebenfalls eine explizite Verschmelzungsoperation nahe.

Etwas allgemeiner ist die *Subsumptionsfaktorisierung*: wenn eine Klausel sich in zwei disjunkte Teilklauseln C und D aufspalten läßt und C subsumiert D mit einer Substitution σ, die keine Wirkung auf D hat, dann können alle Literale der Teilklausel C wegfallen. Diesmal entfällt also nicht der subsumierte, sondern der subsumierende Teil, und es bleibt gerade D übrig. Der Name kommt daher, daß in diesem Fall D ein Faktor der Gesamtklausel $C \cup D$ ist, der seine eigene Elternklausel subsumiert. Das Wegstreichen von C simuliert eine Folge von Faktorisierungs- und Subsumptionsoperationen. Auch hier läßt sich der Subsumptionstest durch einen Test auf Implikation in einer Theorie abschwächen.

Allgemeines Schema für Subsumptionsfaktorisierung:

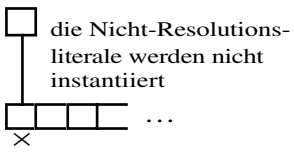


Beispiel: D = {P(a), a≤y, Q(z)}

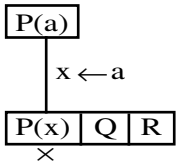


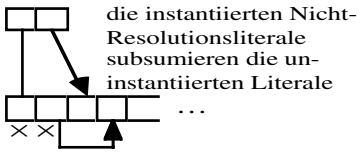
Mit × markierte Literale können aus der Klausel gestrichen werden. Noch allgemeiner ist das Schema der *Subsumptionsresolution*. Es umfaßt alle Fälle, in denen aus einer Klausel durch eine Folge von Resolutions- und Faktorisierungsoperationen eine echte Teilklausel D abgeleitet wird, ohne daß die übrigbleibenden Literale dabei instantiiert werden. In diesem Fall subsumiert D die Elternklausel und alle Zwischenklauseln, so daß die Operation technisch einfach durch Wegstreichen aller Literale außerhalb D realisiert werden kann.

Einige Schemata für Subsumptionsresolution:

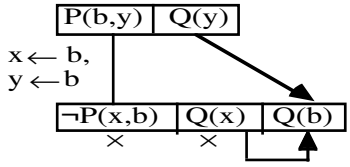


Beispiel: D = {Q, R}





Beispiel: D = {Q(b)}



Die Literallöschungen können im Prinzip beliebig weitergetrieben werden, im Extremfall bis zum Erkennen, daß die ganze Klauselmenge widersprüchlich ist und daher alle Literale einer Klausel entfernt werden dürfen, so daß die leere Klausel übrigbleibt. Dies würde natürlich ein Kriterium erfordern, das so mächtig ist wie das Gesamtverfahren selbst und deshalb den Aufwand nur verlagern. Aber ziemlich viele spezielle Situationen mit Möglichkeiten zur Literallöschung lassen sich durch geschicktes Ausnutzen der Kantenstruktur und der Substitutionen in einem Klauselgraphen effizient erkennen.

Somit ergibt sich ein dritter Vorteil der Klauselgraphrepräsentation: Die Kanten unterstützen eine Vielzahl von Algorithmen zum Erkennen von Redundanzen in der Klauselmenge.

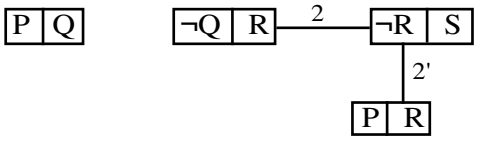
4.2.3 Klauselgraphen mit spezifischen Reduktionsregeln

Die bisher vorgestellte Form der Klauselgraphen und der Operationen darauf sind als computergerechte und effiziente Realisierung des Resolutionskalküls zu bewerten. In diesem Abschnitt werden wir weitere Operationen betrachten, die es erlauben, Kanten oder Klauseln zu entfernen. Dadurch werden Ableitungsmöglichkeiten verboten, die im normalen Resolutionsverfahren logisch noch möglich wären. Damit erhält das Verfahren genau genommen den Status eines neuen Kalküls.

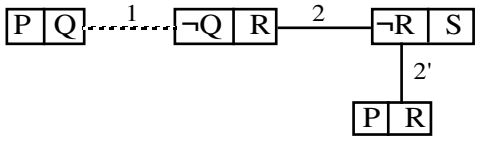
Eine erste Idee ist die Entfernung von *abgearbeiteten* R- oder I-Kanten, um zu verhindern, daß der zugehörige Schritt später ein zweites Mal durchgeführt wird. Diese an sich harmlose Buchhaltungsfunktion erhält zusammen mit dem Kantenvererbungsmechanismus eine ganz neue Qualität, wenn man bedenkt, daß die aus der gelöschten Kante potentiell durch Vererbung entstehenden Kanten und damit alle nachfolgenden Generationen von Resolutionsmöglichkeiten ebenfalls entfallen. Um diesen Effekt zu verdeutlichen, zeigen wir im folgenden Beispiel für einen Anfangsgraphen links eine Folge von Graphen, bei der diese Reduktionsregel angewandt wird, und rechts die analoge Folge von Graphen, bei der die abgearbeiteten Kanten einfach gestrichelt dargestellt sind.

Beispiel:

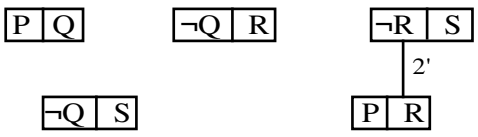
Resolution auf Kante 1 mit Entfernung



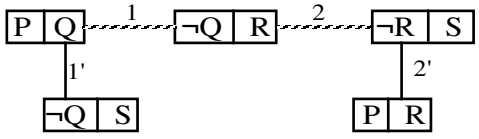
ohne Entfernung



Resolution auf Kante 2 mit Entfernung



ohne Entfernung

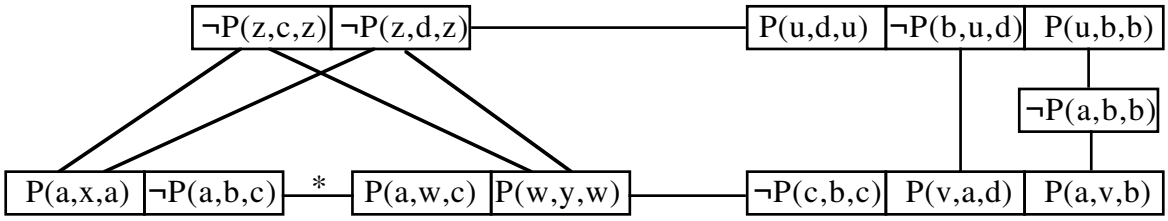


Im linken Graphen ist jetzt nur noch eine Resolutionsmöglichkeit repräsentiert, im rechten dagegen zwei, die jedoch auf dieselbe Resolvente führen.

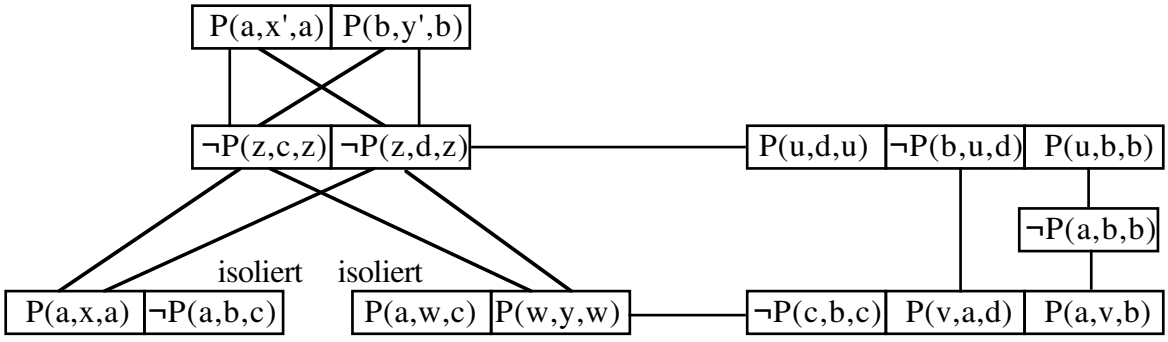
Allgemein bewirkt die Entfernung von abgearbeiteten Kanten, daß eine mehrfache Erzeugung von Resolventen durch verschiedene Resolutionsreihenfolgen aus denselben Klauseln verhindert wird.

Die zweite spezielle Reduktionsregel erlaubt die Entfernung von isolierten Klauseln, das sind solche, die ein isoliertes Literal ohne Kanten enthalten. Diese *Isolationsregel* (englisch *purity*) geht davon aus, daß ein Widerspruch, d.h. die leere Klausel gesucht wird. Literalknoten ohne Kanten können nie „wegresolviert" werden und bleiben immer Bestandteil der Resolventen, die mit der Klausel und deren Nachfolger gebildet werden. Isolierte Klauseln sind daher zur Ableitung der leeren Klausel unbrauchbar und können mit allen ihren Kanten gelöscht werden. Die Entfernung dieser Kanten kann zur Folge haben, daß weitere Literale isoliert werden und deren Klauseln danach ebenfalls entfernt werden können. Eine einzige Reduktion kann daher eine Kettenreaktion von weiteren Reduktionen auslösen.

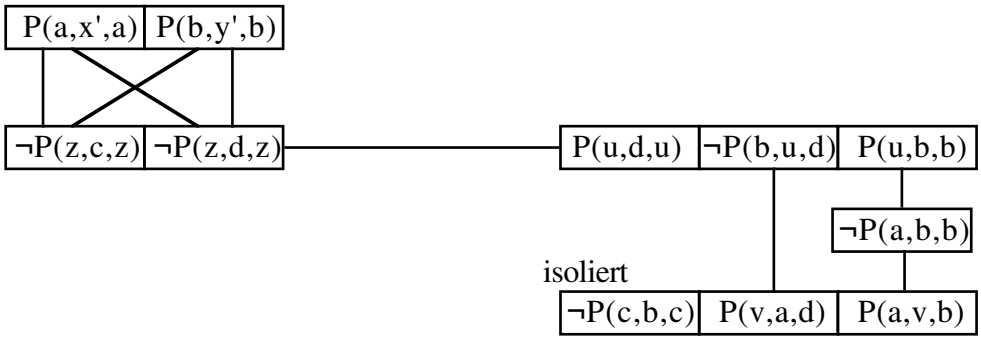
Beispiel:

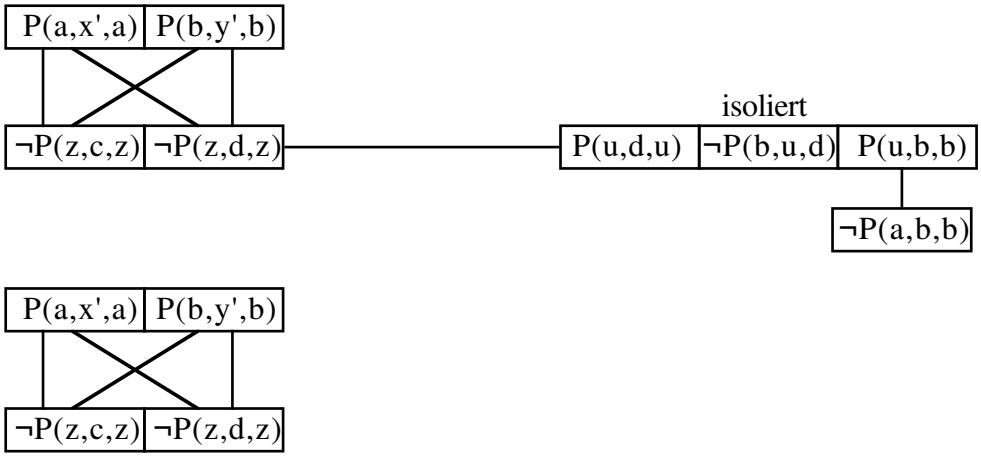


Resolution auf R-Kante * mit Entfernung der abgearbeiteten Kante



Kettenreaktion:



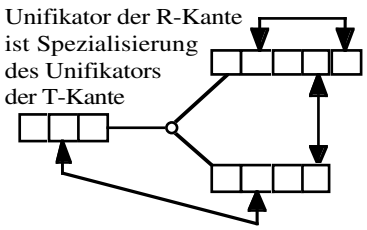


Es ist durchaus möglich, daß alle Klauseln aufgrund der Kettenreaktion gelöscht werden – der Graph *kollabiert* zum leeren Graphen. In diesem Fall war die ursprüngliche Klauselmenge erfüllbar.

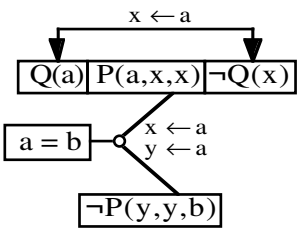
Eine weitere Klasse von speziellen Reduktionsregeln basiert auf der Tatsache, daß jede R- und I-Kante eine neue Klausel repräsentiert. Daher liegt es nahe, diese Kanten schon vor ihrer Ausführung daraufhin zu untersuchen, ob die neue Klausel nicht gleich wieder durch eine der Reduktionsregeln für Klauseln eliminiert würde. In diesem Fall ist es natürlich sinnvoll, die Kante unmittelbar zu entfernen, ohne die Resolvente oder den Faktor erst zu erzeugen. Insbesondere kann eine solche *vorausschauende Kantenlöschung* Klauseln isolieren und damit weitere Reduktionen nach sich ziehen.

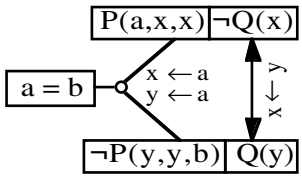
Als Beispiel skizzieren wir die Erkennung von tautologischen R-Kanten, bei der T-Kanten als Indikatoren benutzt werden.

Allgemeines Schema:



Beispiele:





Die Algorithmen zum Erkennen von Kanten, die auf subsumierte oder isolierte Klauseln führen, werden zunehmend komplizierter und trotz der Unterstützung durch I-Kanten als Indikatoren auch rechenaufwendiger. Darüberhinaus muß, um die Vollständigkeit des Gesamtverfahrens zu gewährleisten, nach dem Erkennen einer logisch redundanten Kante geprüft werden, ob sie auch tatsächlich aus dem Graphen gelöscht werden darf, oder ob die durch sie erzeugte Klausel nicht eine der vorher erwähnten Kantenbedingungen verletzen würde. Man muß den Aufwand für das Erkennen einer Reduktionsmöglichkeit gegen den zu erwartenden Nutzen der Reduktion abwägen.

Ein vierter Vorteil des Klauselgraphverfahrens besteht also darin, daß es zusätzliche Reduktionsregeln und damit weitere Einschränkungen des Suchraums unterstützt, die im

normalen Resolutionsverfahren nicht möglich sind. Darüberhinaus bedeutet ein kollabierender Graph – sofern alle verwendeten Reduktionsregeln die Vollständigkeit erhalten –, daß die Ausgangsklauselmenge nicht widersprüchlich ist. Im Hinblick auf die Fähigkeit zum Erkennen der Erfüllbarkeit beim Versuch, die Unerfüllbarkeit zu beweisen, ist die Klauselgraphmethode damit eines der stärksten Verfahren.

4.2.4 Graphen als Repräsentation von Beweisen

Klauselgraphen können für völlig verschiedene Zwecke eingesetzt werden. Bisher haben wir sie als reichhaltigere Datenstruktur für die Zustände von Deduktionssystemen kennengelernt. Man kann aber auch in Form spezieller Klauselgraphen darstellen, durch welche Folge von Ableitungsschritten ein Deduktionssystem die leere Klausel, sein Ergebnis, erreicht. Diese Klauselgraphen repräsentieren dann Widerlegungen und damit Beweise.

In der Mathematik versteht man unter einem Beweis üblicherweise eine Folge von Argumenten, die mit der zu beweisenden Formel bzw. mit dem Widerspruch endet. Von diesem sehr einfachen Beweisbegriff haben wir uns mit dem Tableauverfahren schon etwas gelöst. Ein geschlossenes Tableau ist eine baumartige Datenstruktur, deren Zweige eine vollständige Fallunterscheidung repräsentieren. Ein linearer Beweis in der üblichen Notation läßt sich daraus gewinnen, indem man systematisch alle Fälle nacheinander aufschreibt. Die Beobachtung, daß man dabei eine gewisse Freiheit in der Wahl der aktuellen Reihenfolge hat und damit verschiedene linearisierte Beweise möglich sind, zeigt, daß ein Tableau schon eine leicht abstrahierte Form eines Beweises darstellt. Eigentlich steht es für eine ganze Klasse von Beweisen, die sich nur durch unwesentliche Permutationen der Schritte unterscheiden.

In diesem Abschnitt werden wir eine Abstraktion des Beweisbegriffs für Resolutionsbeweise kennenlernen, bei der sogar der Unterschied zwischen positiven und Widerlegungsbeweisen entfällt. Der Idee liegt die Beobachtung zugrunde, daß die Literale in Resolventen ja nichts anderes sind als Instanzen von Literalen, die schon in der Ursprungsklauselmenge vorkommen. Im Prinzip läßt sich damit eine Resolvente als Menge von Zeigern auf Literale in den Ursprungsklauseln zusammen mit einer Substitution darstellen. Aber selbst das ist nicht nötig. Es genügt im Prinzip, die Resolutionsliterale in den Ausgangsklauseln mit einer R-Kante zu markieren. Eine solche Kante besagt dann (anders als im Klauselgraphverfahren), daß die Resolution als schon ausgeführt zu betrachten ist und damit die beiden Resolutionsliterale „wegresolviert“ sind. Alle nicht mit einer Kante verbundenen Literale, bzw. deren Instanzen, sind damit Bestandteil der Resolvente. Wir wollen das an einem Beispiel demonstrieren.

Beispiel: Abraham ist der Vater von Isaak, und Isaak ist der Vater von Jakob. Daraus folgt, daß Abraham der Großvater von Jakob ist. Die Voraussetzungen und die negierte Behauptung in Klauselform sind:

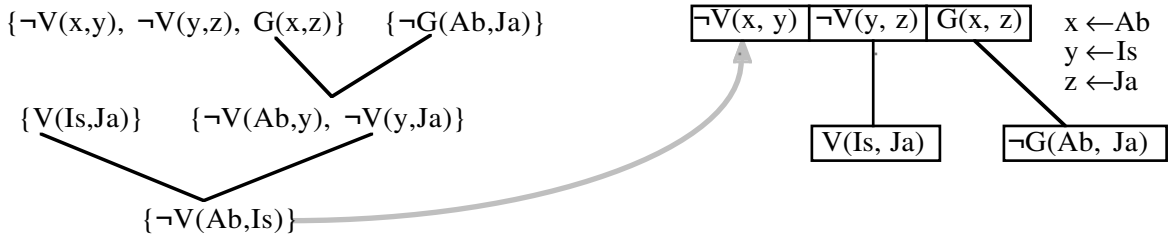
{Vater(Abraham, Isaak)}
{Vater(Isaak, Jakob)}
{¬Vater(x, y), ¬Vater(y, z), Großvater(x, z)}
{¬Großvater(Abraham, Jakob)}

Im folgenden entwickeln wir links eine normale Resolutionswiderlegung und rechts einen Klauselgraphen, in dem nacheinander die Resolutionsschritte durch R-Kanten zwischen den Ausgangsklauseln repräsentiert werden. Die schattierten Verbindungen sind die Zeiger von den Literalen der Resolvente zu ihren Ursprungsliteralen. Aus Platzgründen werden Prädikaten- und Konstantensymbole abgekürzt.

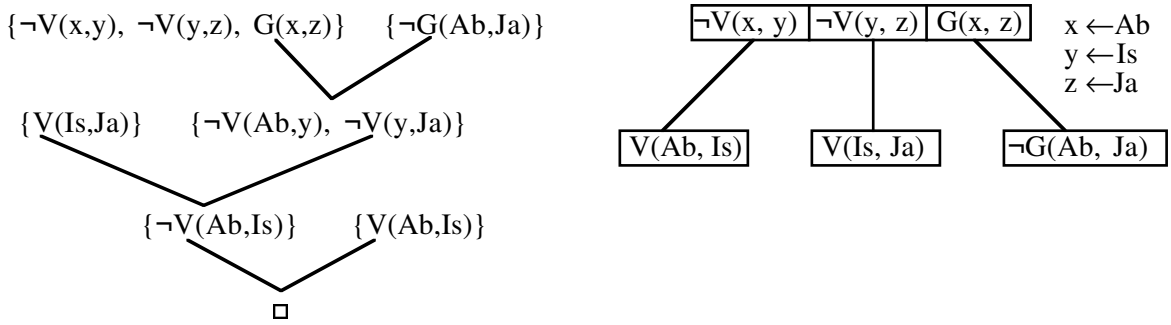
Resolutionswiderlegung zugehöriger Klauselgraph

1. Resolutionsschritt:

2. Resolutionsschritt:

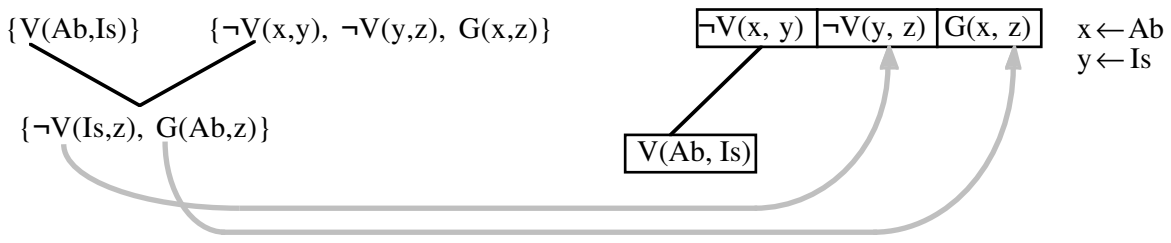


3. Resolutionsschritt:

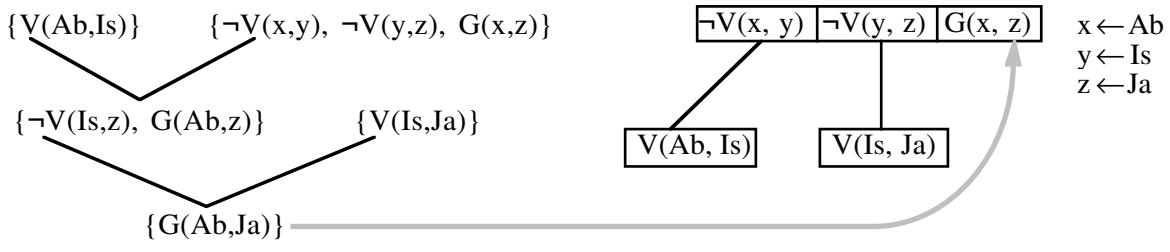


Dies war ein reiner Widerlegungsbeweis. Die nächsten drei Bilder zeigen dagegen einen „positiven“ Resolutionsbeweis, bei dem die Behauptung als vorletzte Klausel explizit abgeleitet wird.

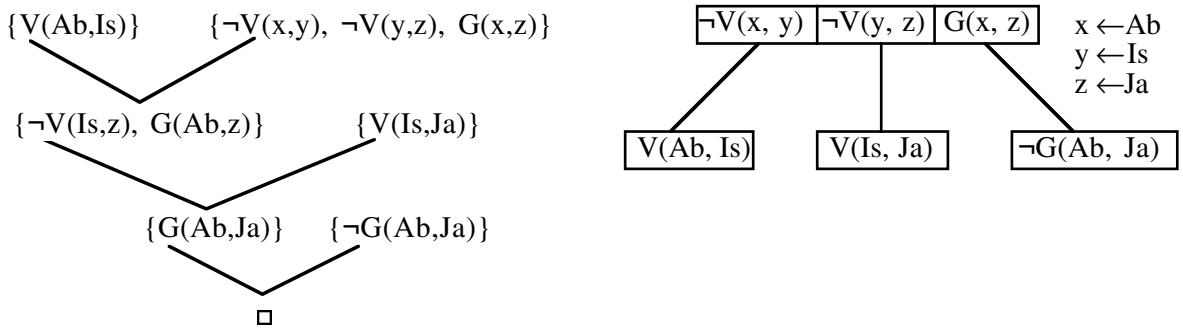
1. Resolutionsschritt:



2. Resolutionsschritt:



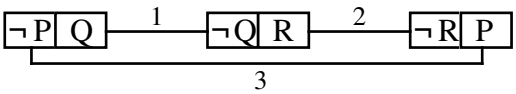
3. Resolutionsschritt:



Obwohl die beiden Resolutionsbeweise verschieden sind, stimmen die jeweils resultierenden Klauselgraphen überein. In beiden Widerlegungen wird jede der drei unären Klauseln genau einmal als Elternklausel in einem Resolutionsschritt verwendet, lediglich die Reihenfolge ist unterschiedlich. Die Klauselgraphdarstellung abstrahiert von dieser unwesentlichen Anordnung der Schritte und repräsentiert damit insgesamt sechs verschiedene Widerlegungen. Man kann die Widerlegungen rekonstruieren, indem man die R-Kanten in beliebiger Reihenfolge mit dem Klauselgraphverfahren abarbeitet. Ein Klauselgraph, der in dieser Weise eine Klasse von Ableitungen der leeren Klausel repräsentiert, heißt ein *Widerlegungsgraph*.

Natürlich ist nicht jeder Klauselgraph auch ein Widerlegungsgraph. Das obige Beispiel zeigt einige Einschränkungen, die solche Graphen erfüllen müssen: jeder Literalknoten ist mit genau einer R-Kante verbunden und es gibt eine Gesamtsubstitution, die jedes Paar von verbundenen Atomen unifiziert. Alle Klauseln stammen aus der ursprünglich zu widerlegenden Klauselmenge. Wenn eine Ursprungsklausel in der Resolutionsableitung zweimal gebraucht wird, enthält der Widerlegungsgraph zwei Kopien der Klausel mit umbenannten Variablen, so daß die Kopien unterschiedlich instantiiert werden können. Eine mehrfach benötigte Resolvente entspricht Kopien des Teilgraphen, der die Resolvente repräsentiert.

Schließlich darf es in einem Widerlegungsgraphen keine Zyklen geben, also Pfade, die von einer Klausel über eine oder mehrere Kanten zu dieser zurückführen. Andernfalls wäre auch der folgende erfüllbare Klauselgraph ein Widerlegungsgraph:

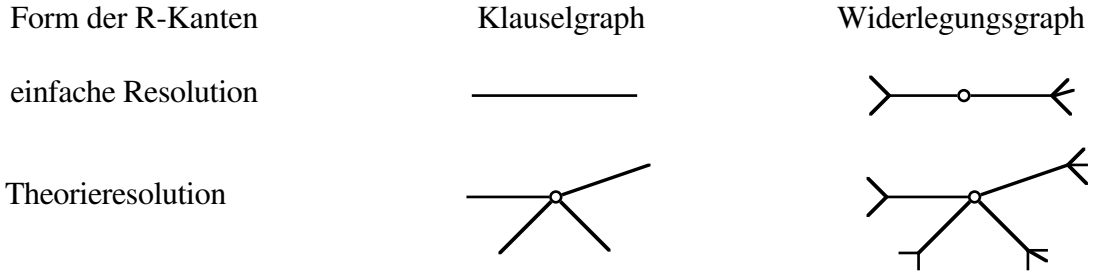


Die Klauselmenge entspricht den Formeln P⇒Q, Q⇒R, R⇒P. Resolution auf den beiden R-Kanten 1 und 2 ergibt die Tautologie {¬P, P}, die leere Klausel kann nicht abgeleitet werden. Die Kantenfolge 1,2,3 ist ein Zyklus. Das Beispiel demonstriert, daß zyklische Pfade

Schlußketten anzeigen, die „sich selbst in den Schwanz beißen“: Um P zu beweisen, muß man R beweisen, um R zu beweisen, muß man Q beweisen, um Q zu beweisen, muß man P beweisen …

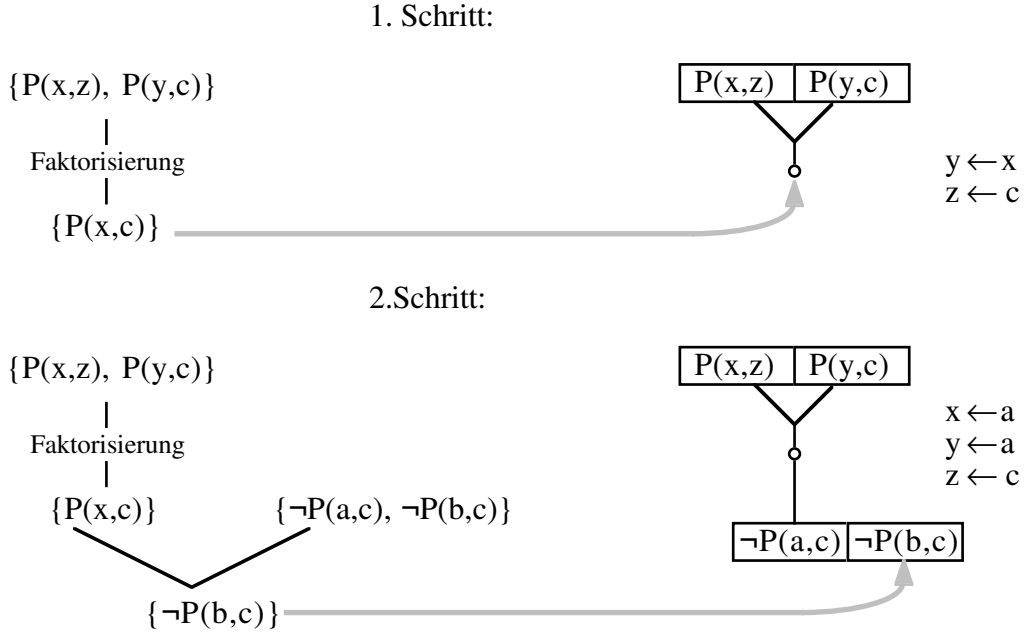
Widerlegungsgraphen wurden zuerst von Robert E. Shostak [Sho76] untersucht. Er konnte zeigen, daß eine Klauselmenge genau dann unerfüllbar ist, wenn es für ausreichend viele Kopien der Klauseln einen Widerlegungsgraphen gibt, d.h. einen nichtleeren, zyklusfreien Klauselgraphen, in dem jeder Literalknoten mit genau einer R-Kante verbunden ist, so daß eine Gesamtsubstitution jedes Paar von verbundenen Atomen unifiziert.

Um in einem Widerlegungsgraphen auch repräsentieren zu können, daß zwei Literale durch Faktorisierung verschmelzen, muß die Form der R-Kanten für Widerlegungsgraphen etwas verallgemeinert werden. Sie dürfen an jedem Ende nicht nur mit einem, sondern mit mehreren Literalknoten verbunden sein und sich zu diesem Zweck gabelförmig verzweigen.

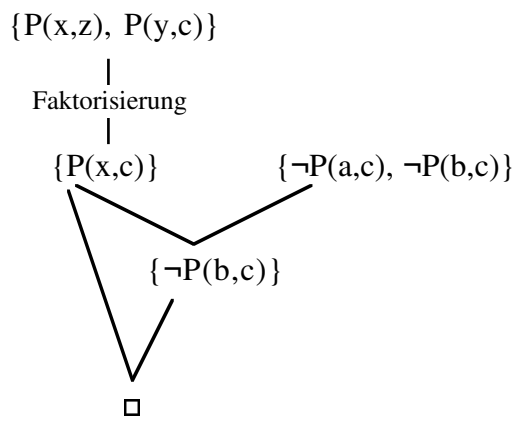


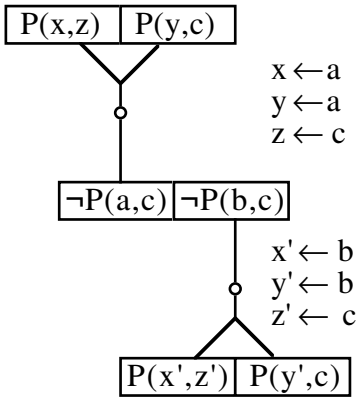
Jede R-Kante besteht aus so vielen Hauptästen, wie Klauseln an dem Resolutionsschritt beteiligt sind – bei einfacher Resolution gerade zwei, bei Theorieresolution mehrere. Jeder Hauptast einer R-Kante fächert in einen oder mehrere Nebenäste aus, die anzeigen, welche Literale verschmelzen müssen, bevor die Resolution tatsächlich durchgeführt werden soll.

Beispiel: Widerlegung Widerlegungsgraph



3.Schritt:



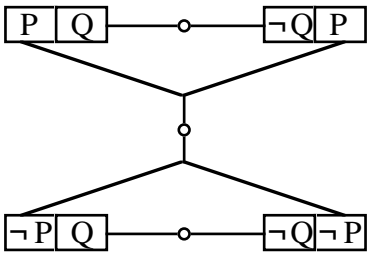


■

Man kann im Prinzip auch für beliebige Klauselgraphen die Definition der R-Kanten, und damit auch der I-Kanten, von denen sie ja nur Spezialfälle sind, in der obigen Weise verallgemeinern. Dann entspricht ein Hauptast einer R-Kante der Disjunktion der Literale an den Nebenästen, die gesamte R-Kante entspricht wie bisher der Konjunktion aller Hauptäste. Für einen Resolutionsschritt muß man aus jedem Hauptast genau einen Nebenast auswählen und die damit verbundenen Literale zusammen als Resolutionsliterale benutzen.

Die Nebenäste müssen nicht alle in dieselbe Klausel führen, wie das in dem letzten Beispiel der Fall ist. Nebenäste, die zu verschiedenen Klauseln führen, zeigen an, daß nicht die Ursprungsklauseln, sondern erst Resolventen faktorisiert werden müssen. Ein Sonderfall liegt vor, wenn für jeden Hauptast alle Nebenäste zur selben Klausel führen. Noch spezieller wird der Widerlegungsgraph, wenn in der Widerlegung keine Faktorisierung vorkommt und somit kein Hauptast einer R-Kante sich in Nebenäste verzweigt. Dann hat der Widerlegungsgraph eine baumartige Struktur und wird auch *Widerlegungsbaum* genannt. Ein Theorem von M. C. Harrison und N. Rubin [HR78] besagt, daß für eine Klauselmenge genau dann ein Widerlegungsbaum existiert, wenn sie unär widerlegbar, also zum Beispiel eine unerfüllbare Hornklauselmenge ist.

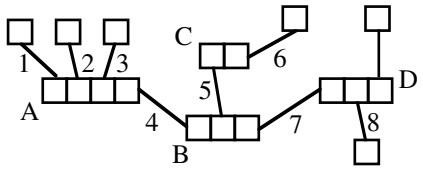
Die Klauselmenge {{P,Q}, {¬P,Q}, {¬Q,P}, {¬Q,¬P}} ist unerfüllbar und besitzt nur Widerlegungsgraphen, die keine Widerlegungsbäume sind. Der Leser mag versuchen, die durch folgenden Widerlegungsgraphen repräsentierten Resolutionswiderlegungen zu konstruieren:



4.2.5 Extraktion von Widerlegungsbäumen aus Klauselgraphen

Die Beschäftigung mit Widerlegungsgraphen gibt eine tiefere Einsicht in die Abhängigkeiten von Schlußfolgerungen voneinander und damit in die topologische Struktur von Resolutionsbeweisen. Widerlegungsgraphen haben sich außerdem als nützlich erwiesen für die theoretische Untersuchung von Eigenschaften von Deduktionssystemen. Ob sie auch bei der Suche nach Beweisen von Nutzen sind, war in den bisherigen Beispielen noch nicht zu erkennen, denn wir haben nur fertige Resolutionswiderlegungen in Widerlegungsgraphen übersetzt. Wie man diese Widerlegungen findet, blieb offen.

In diesem Abschnitt stellen wir ein Verfahren vor, mit dem man für unär widerlegbare Klauselmengen einen Widerlegungsbaum – falls einer existiert – unmittelbar aus einem Klauselgraphen extrahieren kann. Wenn das gelingt, hat man einen Beweis gefunden, ohne eine einzige Resolvente erzeugen zu müssen. Die grundlegende Beobachtung, die zu dem Extraktionsverfahren führt, sieht man am besten an einem etwas komplizierteren Widerlegungsbaum für eine aussagenlogische Klauselmenge:

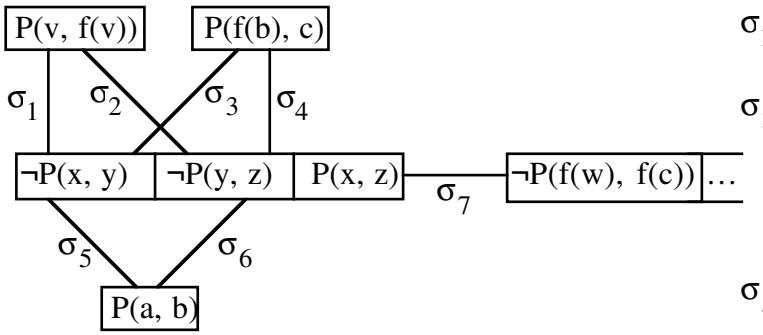


Die Literale
sind weggelassen.

In diesem Widerlegungsbaum gibt es drei Klauseln, A ,C und D, in denen alle Literale bis auf eines mit einer unären Klausel resolvierbar sind. Nimmt man sich eine dieser Klauseln her, beispielsweise A, und resolviert nacheinander mit allen drei unären Klauseln, dann bleibt gerade das vierten Literal A4 übrig, mithin eine neue unäre Klausel, die über einen Nachfolger der Kante 4 mit B verbunden ist. Verfährt man analog mit C, erhält man eine weitere unäre Klausel C1, die über einen Nachfolger von 5 mit B verbunden ist. C1 und A4 zusammen ermöglichen jetzt, die ersten beiden Literale von B wegzuresolvieren, so daß B3 übrigbleibt. Jetzt ist jedes Literal von D mit einer unären Klausel verbunden, und man kann die leere Klausel ableiten.

In einem Klauselgraphen, der alle möglichen R-Kanten enthält, braucht man also einfach nach Teilgraphen zu suchen, in denen sämtliche Literale einer Klausel über eine R-Kante mit unären Klauseln verbunden sind. Gelingt dies, hat man einen Widerlegungsbaum gefunden und ist fertig. Andernfalls betrachtet man die Teilgraphen, in denen alle Literale einer Klausel bis auf eins mit einer unären Klausel verbunden sind. So ein Teilgraph repräsentiert eine unäre Resolvente, die gerade aus einem Nachfolger des einen unverbundenen Literals besteht. Dieses eine Literal kann nun genauso behandelt werden, wie eine „echte“ unäre Klausel, so daß andere Klauseln jetzt einen entsprechenden Teilgraph bilden können, für die das vorher nicht der Fall war.

Bei prädikatenlogischen Klauselmengen müssen zusätzlich die Unifikatoren der R-Kanten berücksichtigt werden. Wie sich das auswirkt, zeigen wir an einem Beispiel, in dem ein Teil des Klauselgraphen ein Transitivitätsaxiom enthält.



$\sigma_1 = \begin{matrix} x \leftarrow v \\ y \leftarrow f(v) \end{matrix}$ $\sigma_2 = \begin{matrix} y \leftarrow v \\ z \leftarrow f(v) \end{matrix}$

$\sigma_3 = \begin{matrix} x \leftarrow f(b) \\ y \leftarrow c \end{matrix}$ $\sigma_4 = \begin{matrix} y \leftarrow f(b) \\ z \leftarrow c \end{matrix}$

$\sigma_7 = \begin{matrix} x \leftarrow f(w) \\ z \leftarrow f(c) \end{matrix}$

$\sigma_5 = \begin{matrix} x \leftarrow a \\ y \leftarrow b \end{matrix}$ $\sigma_6 = \begin{matrix} y \leftarrow a \\ z \leftarrow b \end{matrix}$

In diesem Fall läßt sich das dritte Literal P(x,z) des Transitivitätsaxioms natürlich nicht selbst als neue unäre Klausel ableiten, sondern bestenfalls Instanzen davon. Die möglichen Instanzen werden dadurch bestimmt, daß man kompatible Kombinationen von Unifikatoren der R-Kanten zu den unären Klauseln findet. Um zu sehen, welche das in unserem Beispiel sind, tragen wir die Unifikatoren in einer tabellenartigen Notation auf, in der nur die Einsetzungen für die Variablen x, y und z in einer festen Reihenfolge jeweils untereinander geschrieben werden. Die vier kompatiblen Kombinationen ergeben sich nun, indem man die den Substitutionen durch die Zeilen in den beiden Tabellen zugeordneten Termlisten jeweils paarweise miteinander unifiziert. (Die Variable v in σ_2 muß gegenüber v in σ_1 umbenannt werden, um eine Kopie der Klausel P(v, f(v)) zu simulieren.)

1. Literal:

	x	y	z
σ_1	v	f(v)	z
σ_3	f(b)	c	z
σ_5	a	b	z

2. Literal:

	x	y	z
σ_2	x	v'	f(v')
σ_4	x	f(b)	c
σ_6	x	a	b

kompatible Kombinationen

	x	y	z
σ_1,σ_2	v	f(v)	f(f(v))
σ_1,σ_4	b	f(b)	c
σ_3,σ_2	f(b)	c	f(c)
σ_5,σ_2	a	b	f(b)

Aus den vier kompatiblen Kombinationen lassen sich dann durch Instantiierung des dritten Literals P(x, z) vier neue unäre Klauseln ableiten, nämlich {P(v, f(f(v)))}, {P(b, c)}, {P(f(b), f(c))} und {P(a, f(b))}. Allerdings sieht man an dem Unifikator σ_7, daß nur solche Instanzen sinnvoll weiterverarbeitet werden können, in denen x durch f(…) und z durch f(c) ersetzt werden. Aus dieser Forderung ergibt sich schließlich, daß nur die Kombination σ_3,σ_2 übrigbleibt, die zur unären Klausel P(f(b), f(c)) führt. Die dazu passende Instanz von σ_7 läßt sich jedoch auch mit einem *Verschmelzungsalgorithmus* (englisch *merging*) ohne den Umweg über das instantiierte Literal direkt aus σ_7 und der Kombination σ_3,σ_2 zu {x←f(b), z←f(c), w←b} ausrechnen.

Das Gesamtverfahren zur Extraktion von Widerlegungsbäumen aus Klauselgraphen geht nun so vor:

Man suche eine Klausel, deren Literale alle bis auf höchstens eines, nennen wir es K, mit unären Klauseln verbunden sind. Man berechne aus den Unifikatoren der zugehörigen R-Kanten die Gruppen von kompatiblen Substitutionen. Für jede einzelne R-Kante, die vom Literal K wegführt, erzeuge man mit dem Verschmelzungsalgorithmus die zu den Gruppen passenden Instanzen des zugehörigen Unifikators und markiere die R-Kante damit. Diese instantiierten Unifikatoren entsprechen den Unifikatoren zu den potentiell aus K ableitbaren

unären Klauseln und können in späteren Suchschritten genauso behandelt werden, als ob tatsächlich echte unäre Klauseln vorhanden wären. Der Algorithmus terminiert, wenn eine Klausel gefunden wird, bei der alle Literale mit echten oder potentiellen unären Klauseln verbunden sind und die zugehörigen Substitutionen kompatibel sind. Um den Widerlegungsbaum wirklich zu konstruieren und bei Bedarf zu einem normalen Resolutionsbeweis abarbeiten zu können, muß man lediglich noch alle benutzten Kanten und Substitutionen aufsammeln.

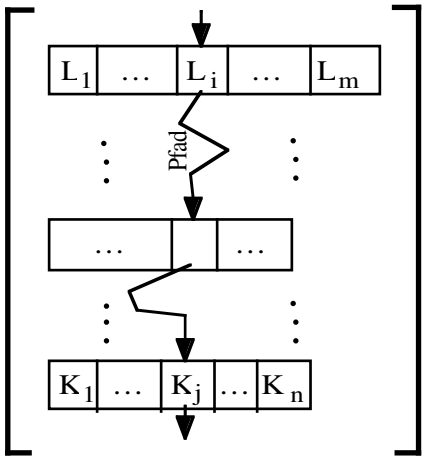
Dieser Ansatz basiert also auf der Erkenntnis, daß sich Widerlegungen charakterisieren lassen durch eine topologische Struktur sowie eine Kompatibilitätseigenschaft der beteiligten Substitutionen. Durch die Extraktion der topologisch geeigneten Teilgraphen werden ganze Klassen von Widerlegungen auf einmal erfaßt, weshalb eine Verbesserung des Gesamtaufwands zu erwarten ist. Zusätzlich bieten sich bei dieser Methode effizienzsteigernde Techniken wie „structure sharing“ besonders an. In der vorgestellten Form ist das Verfahren für Klauselmengen geeignet, die mit der einfachen Resolution unär widerlegbar sind. Die Erweiterung auf Theorieresolution ist unproblematisch. Die Verallgemeinerung auf nicht unär widerlegbare Klauselmengen wurde für die obige Version noch nicht durchgeführt, es gibt aber eine Reihe von Arbeiten mit demselben Grundgedanken [Sic76, CS79, Sho79]. Auch das im nächsten Abschnitt beschriebene Matrixverfahren fällt in diese Kategorie.

4.3 Matrizen

Das *Matrixverfahren*, auch bekannt als *Konnektionsmethode*, verzichtet auf die Erzeugung neuer Formeln [And81, Bib82]. Stattdessen wird die Widersprüchlichkeit der untersuchten Formel direkt aus ihrer inneren Struktur abgeleitet. Um die Grundidee zu erläutern, beschränken wir uns zunächst auf aussagenlogische Formeln in konjunktiver Normalform. Eine derartige Formel hat die Struktur $\mathcal{F} = ((L_1 \vee \ldots \vee L_m) \wedge \ldots \wedge (K_1 \vee \ldots \vee K_n))$, wobei die L_i, K_j negierte oder nicht negierte nullstellige Prädikate sind.

Nach Definition ist $\mathcal{F}$ genau dann erfüllbar, wenn es eine Interpretation gibt, unter der in jeder Klausel mindestens ein Literal wahr ist. Eine Menge von Literalen, die genau ein Element aus jeder Klausel enthält, nennt man einen *Pfad* durch die Klauselmenge. Die Klauselmenge ist also genau dann erfüllbar, wenn es einen Pfad gibt, dessen Elemente unter irgendeiner Interpretation alle erfüllt werden. Die Elemente einer Menge von aussagenlogischen Literalen können genau dann von einer Interpretation erfüllt werden, wenn die Menge kein Paar von komplementären Literalen, also etwa P und ¬P, enthält. Somit ist eine Klauselmenge genau dann erfüllbar, wenn es einen Pfad durch die Klauselmenge gibt, der kein Paar von komplementären Literalen enthält.

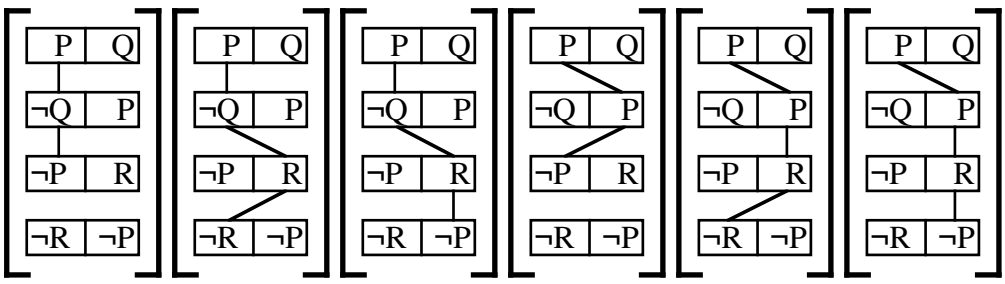
Stellt man die Klauseln so wie im folgenden Bild als Matrix dar, definieren die unter der Interpretation wahren Literale auch graphisch einen „Pfad“ durch die Klauselmenge:

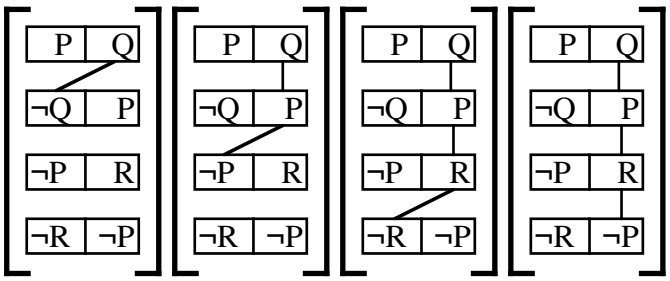


Eine aussagenlogische Klauselmenge ist also genau dann unerfüllbar, wenn jeder mögliche Pfad durch die Menge mindestens ein Paar von komplementären Literalen enthält. Da nur endlich viele Pfade durch eine Klauselmenge existieren, ergibt sich sofort ein Entscheidungsverfahren: Man teste systematisch alle Pfade durch die Klauselmenge daraufhin, ob sie komplementäre Literale enthalten. Findet man nur Pfade mit widersprüchlichen Literalpaaren, dann ist die Formel selbst widersprüchlich, ansonsten ist sie erfüllbar und aus den Literalen eines widerspruchsfreien Pfads ergibt sich sogar ein Modell.

Diese Charakterisierung erlaubt eine elegante Vereinheitlichung von negativen und positiven Testkalkülen. Die obige Matrix repräsentiert in der beschriebenen Weise eine Formel in konjunktiver Normalform. Bei einer dualen Lesweise, bei der die Klauseln disjunktiv und die Literale innerhalb einer Klausel konjunktiv verknüpft sowie negierte Literale als unnegiert und umgekehrt aufgefaßt werden, repräsentiert dieselbe Matrix auch eine (andere) Formel in disjunktiver Normalform. Die beschriebene Eigenschaft der Matrix ist äquivalent zur Unerfüllbarkeit der ersten Formel und zur Allgemeingültigkeit der dualen Formel. Man kann eine beliebige Formel also in konjunktive oder in disjunktive Normalform umwandeln und daraus die jeweilige Matrix herstellen, und dann läßt sich mit demselben Kriterium die Unerfüllbarkeit oder die Allgemeingültigkeit der Formel nachweisen.

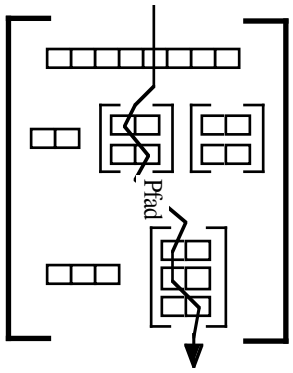
Der Aufwand für den Test aller Pfade durch die Matrix läßt sich verringern. Sobald ein Anfangsstück eines Pfades ein komplementäres Paar von Literalen enthält, gilt dies auch für sämtliche weiterführenden Pfade mit demselben Anfangsstück, und man kann auf deren vollständige Erzeugung verzichten. Daraus ergibt sich eine Art Rücksetzverfahren für die Suche, das wir anhand der unerfüllbaren aussagenlogischen Formel $(P \vee Q) \wedge (\neg Q \vee P) \wedge (\neg P \vee R) \wedge (\neg R \vee \neg P)$ demonstrieren.



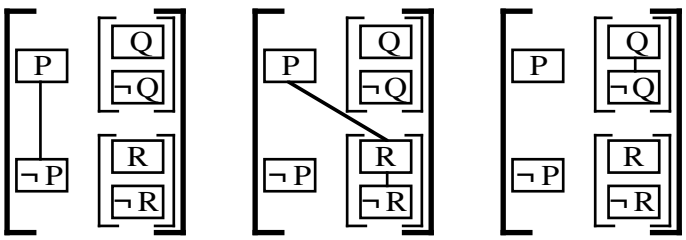


Das Beispiel zeigt auch, daß die Anordnung der Klauseln eine wichtige Rolle spielt. Je früher man einen Pfad abbrechen kann, desto weniger Pfade brauchen abgesucht zu werden.

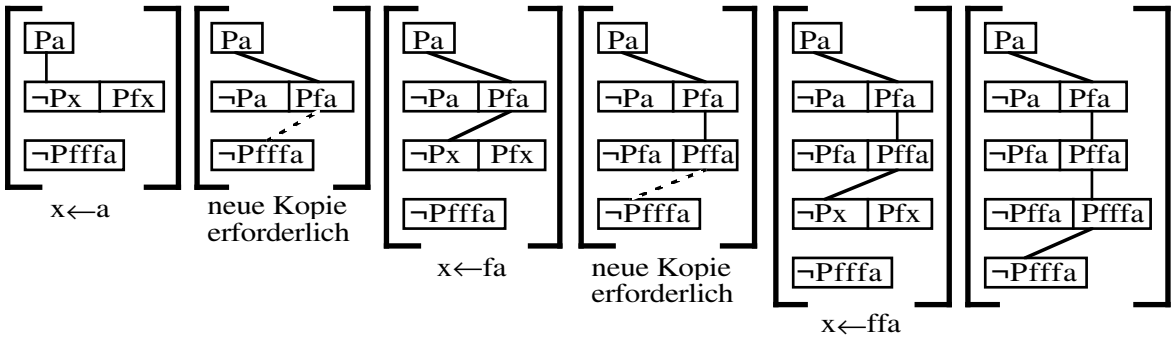
Das Matrixverfahren läßt unmittelbar eine Erweiterung auf Formeln in Negationsnormalform zu. Solche Formeln lassen sich auch in der Form $\mathcal{F} = ((L_1 \vee \ldots \vee L_m) \wedge \ldots \wedge (K_1 \vee \ldots \vee K_n))$ schreiben, wobei die L_i, K_j jedoch wieder Konjunktionen sein können und somit dieselbe Struktur wie $\mathcal{F}$ selbst haben. Der Einfachheit halber bezeichnen wir die L_i, K_j nach wie vor als Literale und die Disjunktionen als Klauseln. Eine solche Formel ist erfüllbar, wenn es eine Interpretation gibt, unter der jede Klausel ein wahres Literal enthält, und diese Bedingung muß rekursiv auf nichtelementare Literale angewandt werden. In der graphischen Darstellung werden nichtelementare Literale durch Submatrizen repräsentiert. Die unter der Interpretation wahren Literale bilden wieder einen normalen Pfad durch die Klauselmenge, wobei der Durchgang durch ein nichtelementares Literal jedoch einem Pfad durch die entsprechende Submatrix entspricht. Das folgende Bild zeigt einen solchen Pfad für eine Formel bestehend aus drei Klauseln, wobei die zweite zwei Submatrizen und die dritte eine Submatrix als Literal enthält.



Jeder Pfad dieser Art muß ein Paar von komplementären Literalen enthalten. Dies mag auf den ersten Blick kompliziert erscheinen, verringert aber oft den Rechenaufwand, wie das obige Beispiel zeigt: Die kompaktere Darstellung von $(P \vee Q) \wedge (\neg Q \vee P) \wedge (\neg P \vee R) \wedge (\neg R \vee \neg P)$ mit Submatrizen entspricht der äquivalenten Formel $(P \vee (Q \wedge \neg Q)) \wedge (\neg P \vee (R \wedge \neg R))$. Das Suchverfahren benötigt jetzt nur drei Schritte:



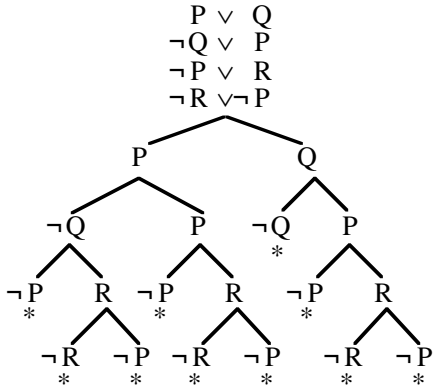
Die Matrixmethode bildet für PL0 ein Entscheidungsverfahren für die Unerfüllbarkeit einer Formel. Für PL1 ist dies nicht möglich, daher muß ein weiteres Element in das Suchverfahren kommen. Es kommt dadurch zustande, daß zwei Literale auf einem Pfad unter Umständen erst durch Instantiierung komplementär gemacht werden müssen. Das bedeutet, daß nicht die Originalklauseln, sondern die mit den entsprechenden Unifikatoren instantiierten Klauseln in der Matrix auftauchen. Wieviele Instanzen von jeder Klausel benötigt werden, kann man zu Beginn nicht wissen, und daher gibt es im erfüllbaren Fall kein allgemeines Abbruchkriterium. Wir zeigen das Vefahren für den prädikatenlogischen Fall am Beispiel der unerfüllbaren Klauselmenge $\{\{P(a)\}, \{\neg P(x), P(f(x))\}, \{\neg P(f(f(f(a))))\}\}$. Aus Platzgründen lassen wir in diesem Beispiel alle Klammern weg.



Die Suche würde nicht terminieren, stünde in der letzten Klausel ein b anstelle eines a. In der Praxis wird man nicht wirklich Einsetzungen der Klauseln erzeugen, sondern die Unifikatoren als Indikatoren für die Einsetzungen geschickt verwalten. Das eigentliche Problem ist jedoch, daß in den meisten Fällen überhaupt nicht klar ist, welche Klausel kopiert werden muß. Das muß dann nach heuristischen Kriterien entschieden werden und kann in ungünstigen Situationen den Beweis erheblich verlängern.

4.3.1 Zusammenhang zwischen Matrix- und Tableauverfahren

Zwischen beiden Verfahren besteht ein sehr einfacher Zusammenhang: Ist die zu beweisende Formel in Klauselform, so entspricht jedem Pfad durch die Matrix genau ein Zweig des zugehörigen Tableaus. Ein Tableaubeweis für das anfangs benutzte Beispiel $(P\vee Q)\wedge(\neg Q\vee P)\wedge(\neg P\vee R)\wedge(\neg R\vee\neg P)$ zeigt diese Korrespondenz ganz deutlich:

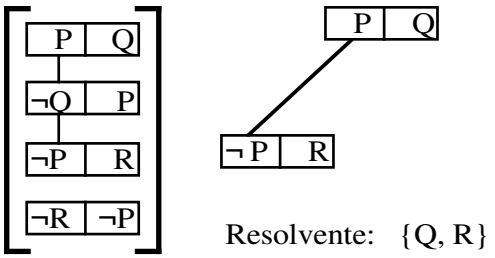


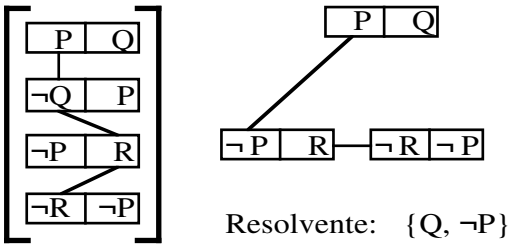
Man kann das Matrixverfahren im Prinzip als eine andere Repräsentation des Tableauverfahrens ansehen. Seine Vorzüge liegen darin, durch Unifikation die Instantiierungsregeln gezielt anwenden zu können, sowie durch die Tatsache, daß keine neuen Formeln erzeugt werden, die Datenstrukturen im Rechner kompakter halten zu können.

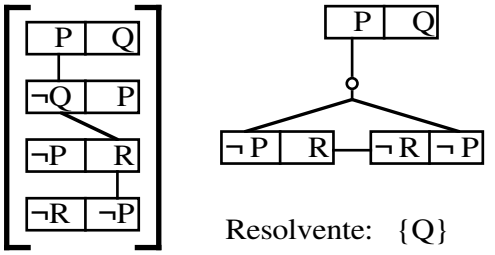
4.3.2 Zusammenhang zwischen Matrix- und Resolutionsverfahren

Das Auffinden von komplementären Literalen auf einem Pfad ist die zentrale Operation im Matrixverfahren. Solche Literalpaare eignen sich als Resolutionsliterale für einen Resolutionsschritt. Deshalb könnte man zunächst vermuten, daß sich die Folge von aufgefundenen komplementären Literalen direkt in eine Resolutionsableitung übersetzen läßt. Das stimmt nur im Prinzip, denn im Matrixverfahren ist die Reihenfolge, in der die Literale gefunden werden, eigentlich unerheblich. Daher enthält ein momentaner Suchzustand, in dem einige Literalpaare gefunden sind und einige noch nicht, Information über eine ganze Klasse von möglichen Resolutionsreihenfolgen und entspricht damit eher einem (teilweise aufgebauten) Widerlegungsgraphen als einer einzelnen Resolutionsableitung.

Wir zeigen die Korrespondenz mit dem schon mehrfach benutzten Beispiel $(P \vee Q) \wedge (\neg Q \vee P) \wedge (\neg P \vee R) \wedge (\neg R \vee \neg P)$. Im folgenden sind noch einmal die Suchzustände des Matrixverfahrens dargestellt, jeweils daneben der entsprechende Teil-Widerlegungsgraph und die zugehörige Gesamtresolvente:

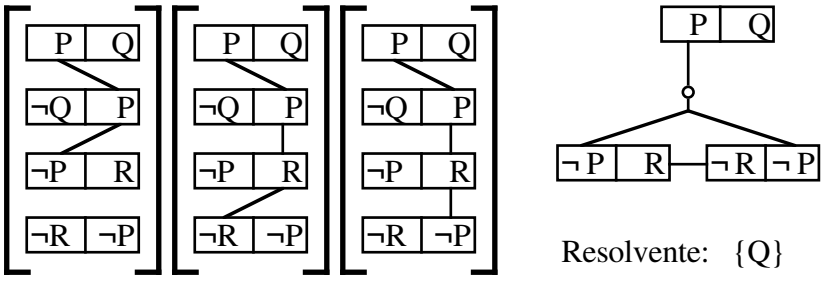


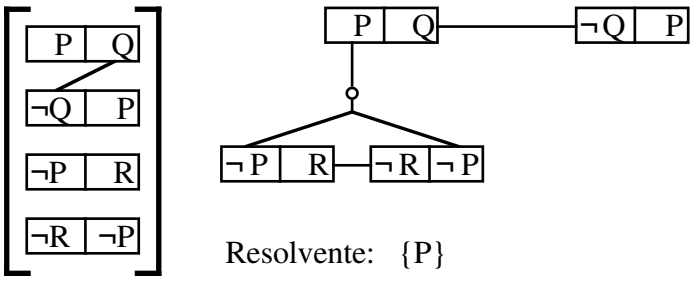


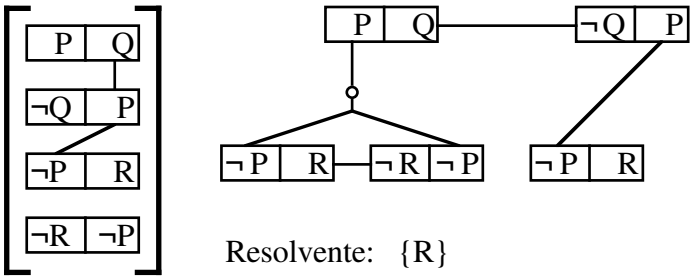


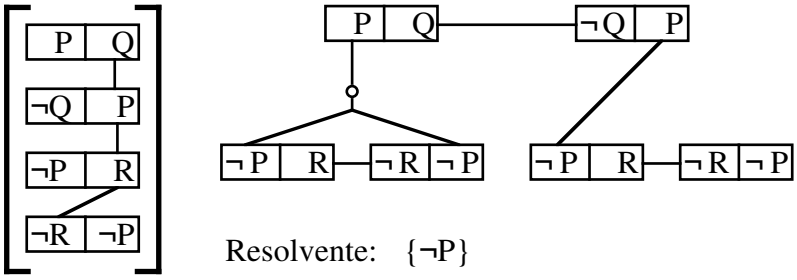
Dies ist ein Fall, in dem sich die Nebenäste eines Hauptastes einer R-Kante (siehe Abschnitt 4.2.4) in zwei Klauseln verzweigen. Die einfachste durch den Graphen repräsentierte

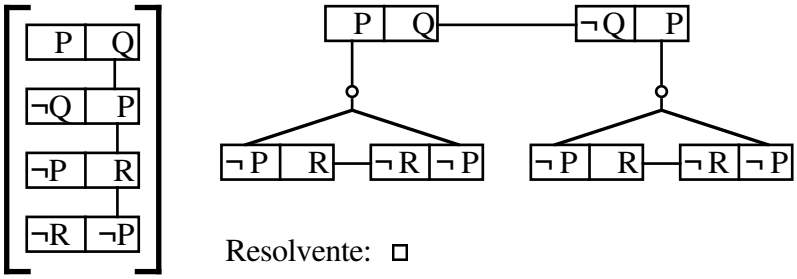
Resolutionsableitung würde zuerst zwischen R und ¬R resolvieren und dann die beiden Exemplare von ¬P verschmelzen, um daraus zusammen mit der ersten Klausel {Q} abzuleiten. Die nächsten drei Schritte haben keinen Einfluß auf den Widerlegungsgraphen. Dies liegt daran, daß die zweite Klausel an keiner der bisherigen Komplementaritäten beteiligt war. Alle Pfade durch ihr zweites Literal sind deshalb auf dieselbe Weise komplementär, wie es die entsprechenden Pfade durch das erste Literal waren. Es liegt nahe, solche Situationen zu erkennen, so daß man sich die nächsten drei Schritte auch tatsächlich spart.











Das Beispiel zeigt eine gewisse Redundanz des Verfahrens, die sich in der Verdopplung des unteren Teils des Widerlegungsgraphen äußert. Tatsächlich wurde die Komplementarität im korrespondierenden unteren Teil der Matrix auch zweimal überprüft. Es ist allerdings ziemlich offensichtlich, wie man die Redundanz beseitigen kann. Ebenfalls naheliegend ist die Verwendung eines Klauselgraphen als zugrundeliegende Datenstruktur, so daß die R-Kanten die Komplementaritätstests unterstützen. Auch eine Einbeziehung von Reduktionsregeln erscheint dann sinnvoll. Eine Vielzahl ähnlicher Verbesserungen und Erweiterungen wird in [Bib82] ausführlich beschrieben. Die genauen Korrespondenzen zwischen den verschiedenen Verfahren hat Elmar Eder untersucht [Eder91].

4.4 Abstrakte Sichtweise der Repräsentationsschicht

Die in Deduktionssystemen eingesetzten Repräsentationen lassen sich mit dem klassischen Kalkülbegriff nicht mehr adäquat beschreiben, insbesondere, wenn Reduktionsregeln beteiligt sind. Man kann sie aber als allgemeine Zustandsübergangssysteme ansehen. Grundlage ist eine Menge von Zuständen irgendwelcher Art mit einer Übergangsrelation zwischen Zuständen. Gewisse Anfangszustände sind ausgezeichnet, und jeder Formel(menge) $\mathcal{F}$ in der passenden syntaktischen Form ist genau ein solcher Anfangszustand $S_{\mathcal{F}}$ zugeordnet (jedenfalls für die auf Testkalkülen basierenden Verfahren, und das sind praktisch alle). Weiter sind noch Klassen von Endzuständen definiert, und jede dieser Klassen steht für eine semantische Eigenschaft wie Unerfüllbarkeit oder Erfüllbarkeit. Ein derartiges System bezeichnen wir als *logisches Zustandsübergangssystem*.

Für eine Formel(menge) $\mathcal{F}$ führt ein logisches Zustandsübergangssystem also ausgehend vom Anfangszustand $S_{\mathcal{F}}$ Zustandsübergänge durch, bis ein Endzustand erreicht ist. Je nach der Klasse dieses Endzustands wird $\mathcal{F}$ damit eine der semantischen Eigenschaften zugesprochen.

In der Trivialrepräsentation für den Resolutionskalkül sind die Zustände einfach Klauselmengen. Übergänge erfolgen von $\mathcal{M}$ nach $\mathcal{M} \cup \{C\}$, wobei C ein Faktor oder eine Resolvente von Klauseln in $\mathcal{M}$ ist. Jede Klauselmenge ist ein zulässiger Anfangszustand. Eine Klasse von Endzuständen steht für Unerfüllbarkeit und besteht aus allen Klauselmengen, die □ enthalten. Die zweite Klasse steht für Erfüllbarkeit und besteht aus allen Klauselmengen, die gegenüber Faktor- und Resolventenbildung abgeschlossen sind, ohne die leere Klausel zu enthalten.

Für die Klauselgraphresolution sind die Zustände Klauselgraphen. Ein Übergang von einem Graphen G zu einem Graphen G' kann aufgrund einer Faktorisierung oder Resolution mit Kantenvererbung sowie Entfernen der abgearbeiteten Kante erfolgen, oder aufgrund einer Reduktionsregel, die nur Objekte entfernt. Der Nachfolgezustand entsteht somit in keinem Fall durch bloßes Hinzufügen einer Formel zum Vorgänger. Zulässige Anfangszustände sind nur die initialen Klauselgraphen, in denen keine möglichen Kanten fehlen. Nach einigen Schritten kann diese Eigenschaft verletzt sein, so daß also auch Zustände vorkommen, die keine Anfangszustände sein können. Die der Unerfüllbarkeit zugeordnete Klasse von Endzuständen besteht aus allen Graphen, die die leere Klausel enthalten. Die Klasse für die Erfüllbarkeit besteht nur aus dem leeren Klauselgraphen.

Das Tableauverfahren benutzt Tableaus als Zustände. Durch Verlängerung eines noch nicht beendeten Zweigs des Vorgängertableaus T erfolgt ein Übergang zu einem Nachfolgertableau T'. Anfangszustände sind die Tableaus, die nur aus einer Wurzel mit der Ausgangsformel bestehen. Die Unerfüllbarkeits-Klasse der Endzustände besteht aus allen Tableaus, deren Zweige sämtlich geschlossen sind. Die Tableaus, in denen zwar alle Zweige beendet, aber einige davon offen sind, bilden die Klasse der Endzustände für die Erfüllbarkeit.

Beim Matrixverfahren ist ein Zustand eine Matrix von Literalen zusammen mit geeigneter Verwaltungsinformation für die jeweils schon erledigten und die noch zu untersuchenden Pfade durch die Matrix. Ein Zustandsübergang führt also zu einem Nachfolgerzustand, in dem sich normalerweise nur die Verwaltungsinformation gegenüber dem Vorgänger geändert hat. Lediglich in den Fällen, in denen eine weitere Kopie einer Klausel erzeugt wird, ändert sich auch die Matrix. Die Anfangszustände sind somit Matrizen mit der Verwaltungsinformation für nur unerledigte Pfade, die Endzustände sind Matrizen mit Verwaltungsinformation, die besagt, daß alle Pfade als komplementär nachgewiesen wurden. Diese Endzustände bilden nur eine Klasse, und zwar für die Unerfüllbarkeit. Bei einer dualen Lesweise repräsentiert dieselbe Matrix eine andere Formel [Bib82], und die Endzustände stehen für deren Allgemeingültigkeit. In einigen Spezialfällen kann man erkennen, daß keine weitere Klauselkopie zur Komplementarität eines Pfades beitragen kann, womit man auch eine Erfüllbarkeits-Klasse bzw., bei dualer Leseweise, eine Falsifizierbarkeits-Klasse von Endzuständen definieren könnte.

In diesem System weichen die Zustände und die Zustandsübergänge also recht stark von den Formeln und den Schlußregeln eines Kalküls ab. Im allgemeinen ist es nicht einmal unbedingt erforderlich, daß die Zustände außer den Anfangszuständen überhaupt noch als logische Formeln gelesen werden können oder daß die Zustandsübergänge Schlußregeln eines Kalküls entsprechen.

Es ist zu erwarten, daß auch die Kalküleigenschaften wie Korrektheit und Vollständigkeit für logische Zustandsübergangssysteme eine etwas andere Gestalt annehmen. Die Unerfüllbarkeits-Korrektheit (auch *Widerlegungskorrektheit* eines logischen Zustandsübergangssystems ist beispielsweise folgende Eigenschaft: wenn aus einem Anfangszustand $S_{\mathcal{F}}$ ein Endzustand der Unerfüllbarkeitsklasse erreichbar ist, dann ist $\mathcal{F}$ eine unerfüllbare Formel. Analog definiert man auch die Korrektheit bezüglich der anderen semantischen Klassen. In diesem Sinn ist das Trivialsystem für den Resolutionskalkül also korrekt bezüglich der Unerfüllbarkeit und auch bezüglich der Erfüllbarkeit: wenn eine Klauselmenge erreicht wird, die die leere Klausel enthält, so ist die Ausgangsmenge in jedem Fall unerfüllbar; wenn eine Klauselmenge erreicht wird, die gegenüber Faktorisierung und Resolution abgeschlossen ist ohne □ zu enthalten, dann ist die Ausgangsmenge in jedem Fall erfüllbar.

Im Gegensatz zu Kalkülen läßt sich für logische Zustandsübergangssysteme die Korrektheit im allgemeinen nicht auf eine „Korrektheit“ der einzelnen Zustandsübergänge von S nach S' zurückführen. Dann müßten nämlich beide Zustände als Formeln gelesen werden können, zwischen denen eine semantische Beziehung besteht (z.B. daß eine genau dann erfüllbar ist, wenn es die andere ist), was nicht immer möglich ist. Außerdem spielt ein Übergang von S

nach S' nur dann eine Rolle, wenn S überhaupt von Anfangszuständen aus erreichbar ist. In der Klauselgraphresolution gibt es beispielsweise unerreichbare Zustände, die unerfüllbare Formeln repräsentieren, von denen aber Übergänge zu erfüllbaren Zuständen möglich sind. Derartige Phänomene im unerreichbaren Teilsystem sind aber irrelevant für die Korrektheit des Systems im obigen Sinn.

Die *Unerfüllbarkeits-Vollständigkeit* (auch *Widerlegungsvollständigkeit*) eines logischen Zustandsübergangssystems ist in naheliegender Weise als die Umkehrung der Korrektheit definiert: wenn $\mathcal{F}$ eine unerfüllbare Formel ist, dann ist aus dem Anfangszustand $S_{\mathcal{F}}$ ein Endzustand der Unerfüllbarkeitsklasse erreichbar. Analog kann man auch die Vollständigkeit bezüglich der anderen semantischen Klassen definieren. Das Trivialsystem für den Resolutionskalkül ist beispielsweise für aussagenlogische Formeln vollständig bezüglich der Unerfüllbarkeit und bezüglich der Erfüllbarkeit. Diese beiden Vollständigkeitseigenschaften kann jedes System natürlich nur bei Einschränkung auf entscheidbare Formelklassen besitzen, jedoch nicht für die volle Prädikatenlogik. Man könnte aber Systeme konstruieren, die ohne Einschränkung vollständig bezüglich der Unerfüllbarkeit und bezüglich der Allgemeingültigkeit sind.

Für den praktischen Einsatz ist noch eine dritte Eigenschaft von Bedeutung. Wenn von einem Zustand S aus Übergänge zu zwei verschiedenen Nachfolgezuständen S_1 und S_2 möglich sind, so kann es passieren, daß von S_1 ein Endzustand der gewünschten Klasse erreichbar ist, von S_2 aber nicht. Die Wahl des Übergangs nach S_2 würde also in eine Sackgasse führen. Um mit solchen Sackgassen im Suchraum umgehen zu können, müßte das System in jedem Fall mit einer *revidierenden Steuerung* (englisch *tentative control regime*, [Nil80]) versehen werden. Diese würde bei Ausführung eines Übergangs Vorkehrungen treffen, später auch die Alternativen berücksichtigen zu können, beispielsweise durch ein Rücksetzverfahren (englisch *backtracking*) oder durch allgemeine Graphsuche oder mit dem A*-Algorithmus.

Eine revidierende Steuerung verwaltet in der einen oder anderen Form letztlich Mengen von Zuständen. Im Fall von Deduktionssystemen ist ein Einzelzustand aber eine sehr große Datenstruktur, zum Beispiel eine Klauselmenge oder ein Klauselgraph oder ein Tableau. Jedenfalls ist die Verwaltung von mehr als einem Zustand zu einem Zeitpunkt nicht praktikabel. Man ist also auf eine *nicht-revidierende Steuerung* angewiesen (englisch *irrevocable control regime*, [Nil80]), die jeweils nur einen Zustand behandelt und bei einem Zustandsübergang sowohl den Vorgänger als auch Alternativen zum Nachfolger „vergißt“. In einer Implementierung könnte der Nachfolgezustand also durch destruktive Manipulation des Vorgängers erzeugt werden.

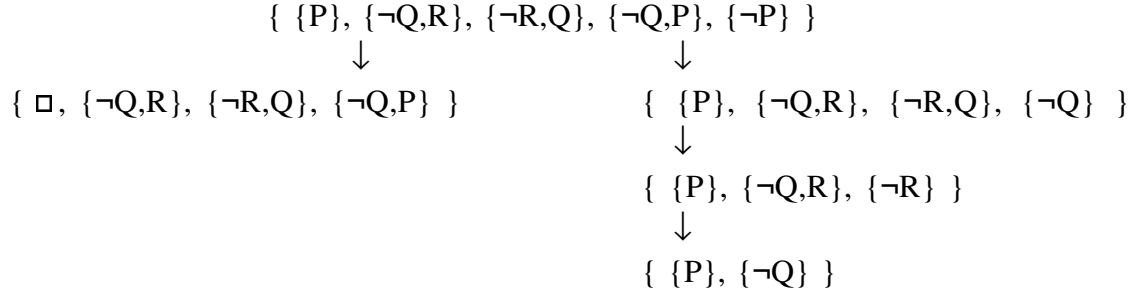
Voraussetzung dafür ist aber, daß der Suchraum keine der oben erwähnten Sackgassen enthält. Diese Bedingung ist im wesentlichen äquivalent zu der bekannten *Konfluenzeigenschaft*: Wann immer aus einem Zustand S zwei Zustände S_1 und S_2 erreichbar sind, so muß es einen Zustand S' geben, der von beiden aus erreicht werden kann. Dabei identifiziert man formal alle Endzustände derselben Klasse und nimmt an, daß von Endzuständen aus keine weiteren Übergänge erlaubt sind. Für ein konfluentes System kann man eine nicht-revidierende Steuerung verwenden. Wird in einem Zustand S der Nachfolger S_2 gewählt und die Alternative

S_1 ignoriert, von der ein Endzustand S' erreichbar wäre, dann ist S' auch von dem gewählten Nachfolger aus erreichbar. Die Wahl eines ungeschickten Schrittes kann die günstigen Schritte also nur hinauszögern, aber nicht gänzlich ausschließen.

Die Konfluenz wurde lange nicht als wichtige Eigenschaft erkannt, da sie vielen Systemen sozusagen automatisch zukommt. Das Trivialsystem für den Resolutionskalkül ist beispielsweise auch auf eine triviale Weise konfluent. Angenommen, von einer Klauselmenge $\mathcal{M}$ sind alternativ Übergänge nach $\mathcal{M} \cup \{C\}$ oder $\mathcal{M} \cup \{D\}$ möglich, wobei C und D Faktoren oder Resolventen von Klauseln in $\mathcal{M}$ sind. Dann ist natürlich von beiden aus auch ein Übergang nach $\mathcal{M} \cup \{C, D\}$ möglich. Das Hinzufügen einer Resolvente wird schließlich in keinster Weise davon betroffen, ob vorher eine andere Resolvente hinzugefügt wurde. Dies ist also ein *kommutatives* System [Nil80], in dem anwendbare Regeln einander nicht behindern können und deshalb unabhängig von der Reihenfolge der Anwendung zum selben Zustand führen. Die Kommutativität ist die einfachste Form der Konfluenz.

Enthält ein System auch Reduktionsregeln, ist die Konfluenz nicht mehr so offensichtlich. Wenn verschiedene Nachfolgerzustände sich strukturell so drastisch unterscheiden können wie im Tableau- oder im Klauselgraphverfahren, kann die Konfluenz sogar ausgesprochen schwer nachzuweisen sein. Auch hier tritt wieder das Phänomen auf, daß man eigentlich nicht an Eigenschaften des gesamten Systems inklusive der unerreichbaren Zustände interessiert ist, weshalb man beispielsweise auf die *Unerfüllbarkeits-Konfluenz* (auch *Widerlegungskonfluenz*) einschränkt: wenn $\mathcal{F}$ eine unerfüllbare Formel ist und aus dem Anfangszustand $S_{\mathcal{F}}$ zwei Zustände S_1 und S_2 erreichbar sind, dann gibt es einen Zustand S', der von S_1 und von S_2 aus erreicht werden kann. Analog kann man wieder die Konfluenz bezüglich der anderen semantischen Klassen definieren. Die Klauselgraphresolution ist z.B. konfluent bezüglich der Unerfüllbarkeit, aber nicht konfluent bezüglich der Erfüllbarkeit.

Zum Abschluß wollen wir noch demonstrieren, daß die Vollständigkeit und die Konfluenz tatsächlich voneinander unabhängig sind. Für aussagenlogische Hornklauselmengen gilt, wenn die leere Klausel überhaupt ableitbar ist, dann auch derart, daß mit keiner Klausel mehr als einmal resolviert wird. Wir modifizieren nun die Trivialrepräsentation für den Resolutionskalkül so, daß bei einem Übergang zwei resolvierbare Klauseln durch ihre Resolvente ersetzt werden, die Elternklauseln also nicht mehr im Nachfolgerzustand enthalten sind. Das resultierende logische Zustandsübergangssystem ist somit für aussagenlogische Hornklauselmengen widerlegungsvollständig. Das folgende Beispiel zeigt, daß es aber nicht widerlegungskonfluent für diese Formelklasse ist:



Die linke Klauselmenge ist ein Unerfüllbarkeits-Endzustand, von der rechten ist gar kein

weiterer Übergang möglich, und die beiden haben keinen gemeinsamen Nachfolger. Im Suchraum gibt es eine Klauselmenge, die die leere Klausel enthält, aber eine nicht-revidierende Steuerung könnte diese verpassen und in eine Sackgasse geraten.

5 Steuerung

Um mit einem logischen Zustandsübergangssystem Beweise zu führen, braucht man – entsprechende Eigenschaften des Systems vorausgesetzt – im Prinzip nur die vom jeweiligen Anfangszustand aus erreichbaren Zustände systematisch aufzuzählen. Praktikabel wird ein Deduktionssystem aber erst dann, wenn es „schlechte“ Schritte möglichst vermeidet und „gute“ Schritte möglichst bevorzugt.

Eine Auswahl der „guten“ Schritte erfordert eigentlich bereichsspezifisches Wissen über das Gebiet, auf das sich die zu beweisenden Aussagen beziehen. Leider existieren bisher kaum Ergebnisse dazu, wie solches Wissen aussieht und wie man ein Deduktionsverfahren damit steuert. Es gibt aber auch rein syntaktische Kriterien, die nur die Struktur der Formeln ausnutzen und deshalb bereichsunabhängig anwendbar sind. Obwohl sie damit natürlich auch entsprechend schwach sein müssen, helfen sie zumindest, die gröbsten Ineffizienzen zu vermeiden. Beim gegenwärtigen Stand der Entwicklung sind diese syntaktischen Kriterien ausschlaggebend dafür, daß Deduktionssysteme überhaupt leidlich funktionieren.

Für den Resolutionskalkül wurden bisher zwei Klassen von syntaktischen Kriterien untersucht. Der Schwerpunkt lag lange Zeit auf den *Restriktionsstrategien*, die unter den überhaupt möglichen Schritten einen Teil gänzlich ausschließen. Damit verringert sich im wesentlichen die Verzweigungsrate im Suchraum, aber dafür erhöht sich oft die Länge der Beweise gegenüber dem unbeschränkten System, so daß der Gesamtnutzen zweifelhaft wird. Manche Restriktionsstrategien haben sich aber empirisch als nützlich erwiesen. Im Gegensatz zu den Restriktionsstrategien schließen *Ordnungsstrategien* keine Schritte aus, sondern bestimmen die Reihenfolge, in der die möglichen Schritte tatsächlich ausgewählt werden. Bei dieser Anordnung können auch syntaktische Heuristiken eine Rolle spielen.

In manchen Fällen wird die Unterscheidung zwischen Restriktions- und Ordnungsstrategien etwas unscharf. Wenn eine Ordnungsstrategie gegenüber gewissen Schritten unendlich viele andere bevorzugt, erreicht sie den Effekt einer Restriktionsstrategie.

5.1 Restriktionsstrategien

Eine der einfachsten Restriktionsstrategien für die Resolution trägt den Namen *unäre Resolution* (englisch *unit resolution*). Sie verbietet die Erzeugung von Resolventen zwischen zwei Elternklauseln, wenn diese beide mehr als ein Literal enthalten. Positiv formuliert muß also jede Resolvente mindestens eine unäre Elternklausel besitzen. Diese Restriktion schränkt die möglichen Nachfolgerzustände einer Klauselmenge stark ein, führt außerdem stets zu Resolventen mit weniger Literalen als sie die längere Elternklausel besitzt und ist obendrein sehr leicht zu implementieren. Auch im praktischen Einsatz hat sie sich recht gut bewährt.

Allerdings ist diese Strategie manchmal zu restriktiv. Eine Restriktionsstrategie sollte möglichst alle Eigenschaften erhalten, die das zugrundeliegende Zustandsübergangssystem besitzt. Während Korrektheitseigenschaften durch eine Restriktionsstrategie nicht beeinträchtigt werden, können Vollständigkeits- und Konfluenzeigenschaften verloren gehen. Die unäre Resolution ist im allgemeinen nicht widerlegungsvollständig, das heißt, die leere Klausel ist mit dieser Strategie nicht von allen unerfüllbaren Klauselmengen aus ableitbar. Immerhin ist die Widerlegungsvollständigkeit für eine wichtige Klasse von Klauselmengen gewährleistet, die die Klasse der Hornklauselmengen umfaßt und die man mangels einer syntaktischen Charakterisierung die *unär widerlegbare* Klasse nennt. Dies ist dieselbe Klasse von Klauselmengen, für die Widerlegungsbäume existieren (siehe Abschnitte 4.2.4 und 4.2.5).

Unabhängig von der Vollständigkeit muß man für Restriktionsstrategien auch die Konfluenz für die relevanten Formelklassen sicherstellen. Man könnte zum Beispiel eine Restriktionsstrategie für die einfache Resolution dadurch definieren, daß man verbietet, daß Klauseln mehr als einmal als Elternklauseln verwendet werden. Für aussagenlogische Hornklauselmengen ist diese Restriktionsstrategie widerlegungsvollständig, aber nicht widerlegungskonfluent. Das oben für logische Zustandsübergangssysteme aufgezeigte Phänomen tritt hier völlig analog wieder auf.

Für die Klasse der unär widerlegbaren Klauselmengen ist auch eine andere wichtige Restriktionsstrategie widerlegungsvollständig, die *Eingaberesolution* (englisch *input resolution*). Diese verbietet die Erzeugung von Resolventen, deren Elternklauseln beide Resolventen sind. Positiv formuliert muß also jede Resolvente wenigstens eine Elternklausel aus der initialen Klauselmenge besitzen. Der wesentliche Vorteil dieser Restriktion besteht daran, daß von allen möglichen Resolutionsschritten ein Resolutionsliteral a priori bekannt ist. Insbesondere treten damit nur Unifikationen zwischen jeweils einem beliebigen und einem a priori bekannten Term auf, so daß man für jeden dieser bekannten Terme aus den initialen Klauseln einen speziellen Unifikationsalgorithmus „kompilieren“ kann. Dieser ist in der Regel wesentlich effizienter als einer, der zwei beliebige Terme unifizieren können muß.

Es gibt eine Reihe von Restriktionsstrategien, die die Grundidee der Eingaberesolution möglichst weitgehend erhalten wollen, aber widerlegungsvollständig für beliebige Klauselmengen sind. Die meisten basieren auf der *linearen Resolution* Dabei werden Resolutionsschritte zwischen zwei Resolventen in den Fällen zugelassen, in denen eine ein „Vorgänger“ der anderen ist. Durch Verschärfung dieser Bedingung sowie durch Beschränkungen für die als Resolutionsliterale zulässigen Literale einer Klausel ergeben sich verschiedene Varianten der linearen Resolution, von denen die sogenannte *SL-Resolution* am erfolgreichsten ist.

Schließlich wird noch die Restriktionsstrategie *Set-of-Support* ziemlich häufig und im großen und ganzen mit Gewinn benutzt. Dazu teilt man die Klauselmenge auf in die Klauseln, die aus den Voraussetzungen entstanden sind, und die Klauseln, die von der negierten Behauptung stammen. Man geht davon aus, daß die Voraussetzungen nicht selbst bereits widersprüchlich sind und deshalb ein Widerspruch nur unter Beteiligung der Behauptung erzielt werden kann. Die Restriktion verbietet deshalb die Erzeugung von Resolventen zwischen zwei

Voraussetzungsklauseln. Etwas allgemeiner kann man die Klauselmenge auch in irgend eine erfüllbare Teilmenge und ihr Komplement aufteilen.

5.2 Ordnungsstrategien

Die simpelste Ordnungsstrategie für die Resolution ist die *Stufensättigungsstrategie*(englisch *llevel saturation*). Dazu ordnet man den Klauseln eine sogenannte Ableitungstiefe zu: Die initialen Klauseln in der Ausgangsmenge haben die Tiefe Null, Faktoren haben dieselbe Tiefe wie ihre Elternklauseln, und die Tiefe einer Resolvente ist um eins größer als die größere der Tiefen der Elternklauseln. Die Stufensättigungsstrategie ordnet die möglichen Schritte einfach gemäß der Ableitungstiefe der jeweils erzeugten Klausel, so daß Klauseln mit einer Tiefe n erst dann in eine Klauselmenge eingefügt werden dürfen, wenn alle Klauseln kleinerer Tiefe bereits abgeleitet sind. Die Reihenfolge der Erzeugung von Klauseln derselben Ableitungstiefe überläßt die Strategie dagegen der Implementierung. Man kann die Bedingung etwa so verschärfen, daß unter den Klauseln derselben Tiefe diejenigen mit weniger Literalen bevorzugt werden.

Eine häufig benutzte Abwandlung der Stufensättigungsstrategie ergibt sich, wenn man die Ableitungstiefe aller Resolventen, die eine unäre Elternklausel besitzen, um eine Konstante k verringert. Damit werden immer noch alle ableitbaren Faktoren und Resolventen systematisch erzeugt, aber solche mit unären Elternklauseln um k Stufen früher als andere. Diese Variante der Stufensättigungsstrategie wurde unter dem Namen *unit preference* bekannt.

Beide Strategien sind zumindest für die Trivialrepräsentation des Resolutionskalküls *erschöpfend* (englisch *exhaustive*): Für jede überhaupt in irgendwelchen Zuständen des Suchraums vorkommende Klausel wird nach endlich vielen Schritten auch ein Zustand erreicht, der diese Klausel enthält (sofern nicht bereits vorher ein Endzustand erreicht worden ist). Wenn das zugrundeliegende System und die benutzte Kombination von Restriktionsstrategien vollständig und konfluent bezüglich einer Klasse von Endzuständen sind, dann wird mit einer erschöpfenden Ordnungsstrategie ein Endzustand der entsprechenden Klasse auch in jedem Fall nach endlich vielen Schritten erreicht. Diese Eigenschaft nennt man auch die *Terminierung* der Ordnungsstrategie bezüglich der jeweiligen Klasse.

In manchen Systemen ist es allerdings gar nicht möglich, daß Ordnungsstrategien erschöpfend sind. Dann muß man wenigstens die *Fairneß* sicherstellen: keinem Schritt dürfen unendlich viele andere vorgezogen werden.

Die Präferierung gewisser Schritte braucht sich nicht auf Resolventen mit unären Elternklauseln zu beschränken, um die Fairneß zu erhalten. Wenn man eine gegebene faire Ordnungsstrategie so modifiziert, daß die Resolutionsschritte Vorrang vor allen anderen haben, deren Resolventen weniger Literale besitzen als beide Elternklauseln, ergibt sich wieder eine faire Ordnungsstrategie. Diese höchste Priorität kann man im Prinzip jeder Klasse von Schritten einräumen, von denen in keinem Zustand unendlich viele hintereinander anwendbar sind, die man aber allgemein für nützlich hält. In der Klauselgraphresolution könnte man etwa die Resolutionsschritte vorziehen, die eine Isolation beider Elternklauseln verursachen, oder die, bei denen eine Elternklausel isoliert wird und die Resolvente kürzer ist als diese.

Auch viele Reduktionsregeln tragen eigentlich zu einer Ordnungsstrategie bei. Die vorausschauenden Kantenlöschungen in der Klauselgraphresolution erzwingen genaugenommen, daß Kanten, deren Faktor oder Resolvente aufgrund einer Reduktionsregel für Klauseln gleich wieder entfernt werden können, mit Vorrang vor allen anderen abgearbeitet werden. Selbst die Subsumptionsfaktorisierung und die Subsumptionsresolution, die ja nur Literale aus einer Klausel entfernen, entsprechen im Grunde der Erzeugung neuer Klauseln. Zieht man sie generell vor, definiert man damit eine neue Ordnungsstrategie. Im allgemeinen lassen sich nicht alle Reduktionsregeln mit allen Ordnungsstrategien kombinieren, ohne die Terminierung oder andere Eigenschaften zu gefährden.

Im Falle der Schritte, die man nicht generell vorziehen darf, kann man eine heuristische Priorität vergeben. Um die Fairneß zu gewährleisten, muß die Ableitungstiefe in diese Priorität eingehen. Man wird aber auch andere syntaktische Größen berücksichtigen, etwa die Anzahl der Literale oder die Komplexität der Terme. Wie allgemein bei heuristischer Suche üblich, berechnet man die Priorität aufgrund einer Gewichtung dieser Werte, die der Benutzer einstellen kann, um so das Verhalten des Systems zu beeinflussen. Im Prinzip könnten die heuristischen Werte auch auf bereichsspezifischem Wissen basieren, obwohl hier der Standardeinwand gilt, daß man solches Wissen kaum durch einen einfachen Prioritätswert codieren kann.

Überhaupt verfügt der Benutzer oft noch über Kontrollwissen, das dem System in geeigneter Weise zugänglich gemacht werden sollte. Mit welchen Mechanismen solches Kontrollwissen eingebracht werden kann und wie es im einzelnen beschaffen ist, ist zur Zeit aber noch weitgehend unerforscht.

Für das Resolutionsverfahren gibt es schließlich noch einige erfolgreiche Strategien, die zwar nicht alle möglichen Schritte in eine Abarbeitungsreihenfolge bringen, aber gewisse Folgen von Schritten so zusammenfassen, daß sie nur gemeinsam oder gar nicht ausgeführt werden können. Aus diesem Grund kann man sie als eine Art von Ordnungsstrategien ansehen.

Bei der *UR-Resolution* (englisch *unit resulting resolution*) [MOW76] wird jeweils eine Klausel mit n+1 Literalen, der sogenannte „Nukleus“, simultan mit n unären Klauseln resolviert, so daß eine neue unäre Klausel entsteht. Seien beispielsweise folgende Klauseln gegeben:

$$C1 = \{\neg P(x, y), \neg P(y, z), P(x, z)\}$$
$$C2 = \{P(a, b)\}$$
$$C3 = \{P(b, c)\}.$$

Damit sind u.a. folgende Resolutionsschritte möglich:

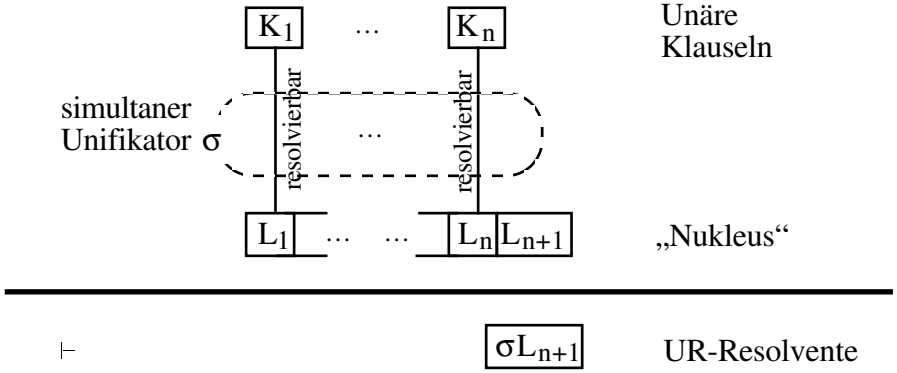
$$C1,1 \;\&\; C2 \vdash \quad R1 = \neg P(b, z), P(a, z)$$
$$R1,1 \;\&\; C3 \vdash \quad R2 = P(a, c).$$

Die zweite Resolvente entsteht aber auch mit einer anderen Ableitung, die sich nur durch eine unwesentliche Vertauschung der Reihenfolge der Schritte von der ersten unterscheidet:

$$C1,2 \;\&\; C3 \vdash \quad R1' = \neg P(x, b), P(x, c)$$
$$R1,1 \;\&\; C2 \vdash \quad R2 = P(a, c).$$

Die UR-Resolution faßt die beiden Schritte so zusammen, daß R2 unmittelbar abgeleitet wird

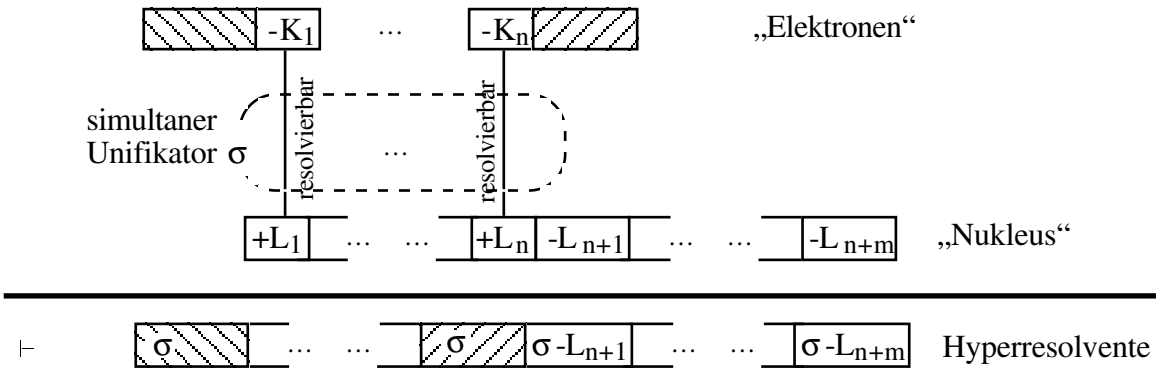
und die Reihenfolge der Schritte keine Rolle mehr spielt. Das allgemeine Schema für UR-Resolution sieht in graphischer Darstellung folgendermaßen aus:



Diese Darstellung verdeutlicht, daß die unären Klauseln in der Ausgangssituation gleichberechtigt sind, und daß daher die Reihenfolge, in der sie bearbeitet werden, keine Auswirkung auf das endgültige Resultat haben sollte. Der gemeinsame Unifikator wird berechnet, indem man durch Aneinanderhängen der Termlisten der unären Klauseln und der Termlisten der Partnerliterale im Nukleus zwei große Termlisten bildet und diese ganz normal unifiziert. Im Transitivitätsbeispiel von oben sind die Termlisten (x, y, y, z) und (a, b, b, c). Deren Unifikator ist {x←a, y←b, z←c}. Es gibt allerdings auch effizientere Methoden.

Die Wirkung der UR-Resolution ist nicht nur, daß von der Reihenfolge der n unären Resolutionsschritte abstrahiert wird. Obendrein werden die als Zwischenergebnisse anfallenden n-1 Klauseln gar nicht erzeugt, im obigen Beispiel würden also R1 oder R1' nicht in die Klauselmenge eingefügt. Da es dafür zunächst keinen Grund gibt, jedenfalls nicht mit irgendeiner gängigen Reduktionsregel, entspricht die UR-Resolution genaugenommen sogar einer ganz neuen Schlußregel, für die wieder dieselben Eigenschaften gezeigt werden müssen wie für die Resolutionsregel. Die UR-Resolution ist widerlegungsvollständig für die unär widerlegbare Klasse von Klauselmengen. Das in Abschnitt 4.2.5 beschriebene Verfahren zur Extraktion von Widerlegungsbäumen simuliert im wesentlichen gerade eine UR-Ableitung.

Die *Hyperresolution* kann man als Verallgemeinerung der UR-Resolution ansehen. Sie wurde von John Alan Robinson entworfen [Rob65b] und wird durch folgendes Schema beschrieben:



Als „Nukleus“ dient eine Klausel, die mindestens ein positives Literal enthält. Solche Klauseln gibt es in unerfüllbaren Klauselmengen immer. Für jedes positive Literal des Nukleus benötigt

man ein sogenanntes „Elektron“, eine Klausel mit nur negativen Literalen. Rein negative Klauseln kommen in unerfüllbaren Klauselmengen ebenfalls immer vor. Der Nukleus wird wieder simultan mit allen Elektronen resolviert, wodurch eine rein negative Klausel entsteht, die wiederum als Elektron im nächsten Hyperresolutionsschritt dienen kann. Die rein negativen Klauseln übernehmen hier also die gleiche Rolle wie die unären Klauseln in der UR-Resolution.

Dual zu dieser sogenannten negativen Hyperresolution definiert man die positive Hyperresolution, bei der die Vorzeichen der Literale in Nukleus und Elektronen gerade vertauscht sind. Da im Normalfall eine negierte Behauptung nur negative Literale enthält und damit als Elektron für negative Hyperresolution verwendbar ist, eignet sich diese für Rückwärtsschließen von der Behauptung in Richtung Voraussetzungen, während positive Hyperresolution gerade umgekehrt von den Voraussetzungen in Richtung auf die Behauptung arbeiten kann. Beide Varianten der Hyperresolution (mit eingebauter Faktorisierung) sind widerlegungsvollständig für beliebige Klauselmengen. Normalerweise kombiniert man die Hyperresolution mit einer fairen Ordnungsstrategie.

6 Zwei konkrete Deduktionssysteme

Als konkrete Deduktionssysteme, in denen viele der angesprochenen Ideen verwirklicht sind, werden das in Deutschland entwickelte MKRP-System sowie der am Argonne National Laboratory entwickelte Beweiser OTTER vorgestellt.

6.1 MKRP

Das an den Universitäten Karlsruhe und Kaiserslautern entwickelte automatische Beweissystem „Markgraf Karl Refutation Procedure (MKRP)“ [OS91] basiert auf dem Klauselgraphverfahren (Markgraf Karl war der Gründer der Stadt Karlsruhe). Seine Funktionsweise verdeutlicht in etwa, wie alle Komponenten eines Deduktionssystems zusammenwirken.

Eingabe: Eine Menge von Voraussetzungen und eine Behauptung, formuliert in einer Sprache für mehrsortige Prädikatenlogik erster Stufe.

Ausgabe: Ein aufbereitetes Protokoll der Widerlegung oder ein Hinweis, daß die Behauptung nicht aus den Voraussetzungen folgt.
(Dies wird in Spezialfällen erkannt, das Verfahren terminiert aber nicht immer)

1. Die Voraussetzungen und die negierte Behauptung werden in Klauselform transformiert. Dabei wird die Behauptung automatisch in möglichst viele unabhängige Teilprobleme zerlegt und für jedes eine eigene Beweissuche angestoßen.
2. Der initiale Klauselgraph wird vollständig aufgebaut.
3. Eine Vielzahl von Klausel-, Literal- und Kantenlöschregeln werden angewandt, um den Graphen zu verkleinern.

4. Die eigentliche Suchschleife iteriert, bis entweder die leere Klausel gefunden wurde, der Graph kollabiert, oder bestimmte Abbruchkriterien erfüllt sind:
   – Auswahl einer oder mehrerer Kanten;
   – Erzeugung der Resolventen und Faktoren;
   – Reduktion des neuen Graphen.

5. Das Protokoll wird aufbereitet.

Die Auswahl der Kanten geschieht nach folgenden Gesichtspunkten: Als erstes werden R-Kanten ausgewählt, deren Abarbeitung zur Folge hat, daß der Graph anschließend durch Reduktionen kleiner wird. Beispielsweise kann das der Fall sein, wenn nach der Löschung der abgearbeiteten Kante eine oder beide Elternklauseln isoliert sind und daher wegfallen. Im allgemeinen wird jedoch die Wirkung der Reduktionsregeln bis zu einem gewissen Grad vorausberechnet. Wenn es keine derartige Kante mehr gibt, wird mit dem in 4.2.5 vorgestellten Verfahren versucht, einen Widerlegungsbaum aus dem Klauselgraphen zu extrahieren. Falls tatsächlich einer gefunden wurde, ist der Beweis beendet, wenn nicht, hat man meist genügend viel Information gesammelt, um zumindest neue unäre Klauseln abzuleiten. Wenn auch das zu keinem Erfolg geführt hat, wird eine Kante aufgrund der vom Benutzer vorher festgelegten Restriktions- und Ordnungsstrategie ausgewählt.

Die Bereitstellung der Informationen, die als Grundlage für die Auswahl der abzuarbeitenden Kante dienen ist der rechenaufwendigste Teil. Daher ist die Zahl der nach außen sichtbaren Deduktionen pro Sekunde relativ gering, das Verhältnis von notwendigen zu nutzlosen Klauseln aber relativ hoch.

6.2 OTTER

OTTER (Organized Techniques for Theorem Proving and Effective Research) wurde von Bill McCune am Argonne National Laboratory in Chicago entwickelt. Der Beweiser ist das letzte Produkt in einer Reihe von Systemen (AURA, ITP), die dort unter Leitung von Larry Wos seit Erfindung des Resolutionsprinzips verwirklicht wurden.

Es handelt sich um einen Resolutionsbeweiser für unsortierte Prädikatenlogik erster Stufe mit Gleichheit. Die eingebauten Ableitungsregeln sind Faktorisierung, Resolution, Hyperresolution, UR-Resolution und Paramodulation. Einige weitere Operationen sind Konvertierung in Klauselform, Vereinfachung von Termen durch gerichtete Gleichungen (Demodulation), Auswahl der Ableitungen über Gewichtsfunktionen, evaluierbare Prädikate und Funktionen, Termordnungen und Knuth-Bendix Vervollständigung (vgl. Kapitel Gleichheitsbehandung). OTTER ist in C geschrieben (ca. 180 KByte Code) und für viele Computer frei erhältlich.

Arbeitsweise von OTTER

Otter hält drei Listen von Klauseln, Axiome, SOS (Set of Support) und gerichtete Gleichungen (Demodulatoren). Alle Anfangsklauseln müssen vom Benutzer einer dieser Listen zugewiesen werden. Zusätzlich kann durch bestimmte Eingaben eine Gewichtung der Klauseln definiert

werden. Die Gewichtung ist eine Funktion, die aus der syntaktischen Form der Klausel einen numerischen Wert berechnet.

Die Hauptkontrollschleife sieht folgendermaßen aus:

> Solange die SOS Liste nicht leer ist und keine leere Klausel gefunden wurde:
>
> 1. Berechne die Klausel in der SOS Liste mit dem kleinsten Gewicht („given_clause“).
> 2. Lösche die given_clause aus der SOS Liste und hänge sie an die Axiome Liste.
> 3. Wende alle eingestellten Deduktionsregeln an und zwar so, daß jeweils die given_clause und Klauseln aus der Axiome Liste beteiltigt sind, d.h. mache alle Inferenzen zwischen der given_clause und den Axiomen. Auf jede abgeleitete Klausel wird eine Prozedur Process-new-clause angewendet. Falls die Klausel danach noch exitiert, wird sie an die SOS Liste angehängt.

Die Prozedur Process-new-clause:
Die neue Klausel C wird folgendermaßen bearbeitet (optional bedeutet, daß die Operation nur durchgeführt wird, wenn eine entsprechendes Flag vom Benutzer gesetzt wurde.)

1. (optional) Drucke C.
2. Vereinfache (demoduliere) C mit den in der Demodulatoren Liste vorhandenen Demodulatoren und evaluiere alle evaluierbaren Terme und Atome. Terme und Atome sind dann evaluierbar wenn eine entsprechende Funktion vom Benutzer eingegeben worden ist und wenn alle Argumente zu Elementen der entsprechenden Trägermenge (i.a. Zahlen) instantiiert wurden. Beispielsweise ist der Term (+ a x) nicht evaluierbar, wohl aber (+ 3 4).
3. (optional) Richte Gleichungen (das wird für die Knuth-Bendix Vervollständigung benötigt).
4. Verschmelze identische Literale.
5. (optional) Sortiere Literale.
6. (optional) Lösche C und stoppe, falls die Anzahl ihrer Literale eine vorgegebene Schranke überschreitet.
7. Lösche C und stoppe, falls C eine Tautologie ist.
8. (optional) Lösche C und stoppe, falls das Gewicht der Klausel eine vorgegebene Schranke überschreitet.
9. (optional) Lösche C und stoppe, falls C von einer Klausel in Axiome oder SOS subsumiert wird (forward subsumption).
10. (optional) Wende unäre Subsumptionsresolution auf C an, d.h. einer der Resolutionspartner muß eine unäre Klausel sein.
11. Hänge C an die SOS Liste an.

12. (optional) Drucke C als „kept clause“.

13. Falls C kein Literal hat, ist die Leere Klausel gefunden.

14. Falls C genau ein Literal hat, such nach einer mit C komplementär unifizierbaren unären Klausel. Falls eine existiert ist ein Widerspruch gefunden.

15. (optional) Falls eine Widerlegung gefunden wurde, drucke den Beweis.

16. (optional) Versuche, C in einen Demodulator umzuwandeln. Das geht wenn C aus einem Gleichheitsliteral besteht, wovon eine Seite in der Termordnung (vom Benutzer vorgegeben) größer ist als die andere Seite.

17. (optional) Falls bei Schritt 16 ein Demodulator erzeugt wurde, demoduliere damit alle vorhanden Klauseln.

18. (optional) Lösche alle Klauseln in Axioms und SOS, die von der neuen Klausel subsumiert werden (backward subsumption).

19. (optional) Faktorisiere die neue Klausel und behandle die Faktoren nach dem gleichen Schema.

Durch besonders trickreiche Implementierung und insbesondere durch Indexingtechniken, mit denen man große Mengen von Termen verwalten kann und schnell für einen gegbenen Term die damit unifizierbaren Terme wiederfindet, ist das Programm sehr effizient und kann große Mengen (Milliarden!) von Klauseln handhaben.

Literatur

And81 P.B. Andrews: *Theorem Proving via General Matings*, J.ACM, Vol. 28 (1981), 193-214.

Bib82 W. Bibel: *Automated Theorem Proving*, Vieweg (1982).

Chu36 A. Church: *A Note on the Entscheidungsproblem*, J. Symbolic Logic, No. 1 (1936), 40-41.

CL73 C.-L. Chang, R.C. Lee: *Symbolic Logic and Mechanical Theorem Proving*, Computer Science and Applied Mathematics Series (Editor W. Rheinboldt), Academic Press, New York (1973).

CS79 C.-L. Chang, J.R. Slagle: *Using Rewriting Rules for Connection Graphs to Prove Theorems*, Artificial Intelligence, Vol. 12, No.2, (1979), 159-178.

Eder91 E. Eder, *Relative Complexities of First Order Calculi*, Vieweg, Braunschweig 1991.

EFT78 H.-D. Ebbinghaus, J. Flum, W. Thomas: *Einführung in die mathematische Logik*, Wissenschafliche Buchgesellschaft, Darmstadt (1978).

EOP91 N. Eisinger, H.J. Ohlbach, A. Präcklein: *Reduction Rules for Resolution Based Systems*, Artificial Intelligence, Vol. 50, No. 2 (1991), 141-181.

Fit83 M.C. Fitting: *Proof Methods for Modal and Intuitionistic Logics*, Reidel, Dordrecht, Holland (1983).

Fit90 M.C. Fitting: *First Order Logic and Automated Theorem Proving*, Springer Verlag 1990.

Fre79 G. Frege: *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*, Nebert Verlag, Halle (1879).

Fro86 R.A. Frost: *Introduction to Knowledge Base Systems*, Collins Professional and Technical Books, London (1986).

Gen35 G. Gentzen: *Untersuchungen über das logische Schließen*, Mathematische Zeitschrift 39 (1933), 176-210.

Göd30 K. Gödel: *Die Vollständigkeit der Axiome des logischen Funktionenkalküls*, Monatshefte für Mathematik und Physik 37 (1930), 349-360.

Göd31 K. Gödel: *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme*, Monatshefte für Mathematik und Physik 38 (1931).

HR78 M.C. Harrison, N. Rubin: *Another Generalization of Resolution*, J.ACM, Vol. 25, No. 3 (1978), 341-351.

Her30a J. Herbrand: *Investigations in proof theory: the properties of the propositions*,

in *From Frege to Gödel: a Source Book in Mathematical Logic* (J. van Heijenoort, Ed.), Harvard Univ. Press., Cambridge, Massachusetts (1967), 618-628.

Her30b J. Herbrand: *Recherches sur la théorie de la démonstration*, (Thèse Paris), Warszawa (1930), Kap. 3;

ebenfalls in *Logical Writings* (W.D. Goldfarb, ed.), Harvard University Press (1971).

Kow70 R. Kowalski: *The Case for Using Equality Axioms in Automatic Demonstration*, Proc. IRIA Symposium on Automatic Demonstration, Versailles, Lecture Notes in Mathematics, Band 125, Springer-Verlag, Berlin (1970), 112-127.

Kow75 R. Kowalski: *A Proof Procedure Using Connection Graphs*, J.ACM Vol. 22, No.4 (1975), 572-595.

LeSc92 R. Letz, J. Schumann, S. Bayerl, W. Bibel: *SETHEO: A High Performance Theorem Prover*. Erscheint im Journal of Automated Reasoning 1992.

Lin69 P. Lindström: *On extensions of elementary logic*, Theoria 35 (1969).

Lov78 D. Loveland: *Automated Theorem Proving: A Logical Basis*, Fundamental Studies in Computer Science, Vol. 6, North-Holland, New York (1978).

Mar84 P. Martin-Löf: *Intuitionistic Type Theory*, Notes by G. Sambin, Bibliopolis Verlag (1984).

MOW76 J. McCharen, R. Overbeek, L. Wos: *Complexity and Related Enhancements for Automated Theorem-Proving Programs*, Computers and Mathematics with Applications, Vol. 2 (1976), 1-16.

Nil80 N. Nilsson: *Principles of Artificial Intelligence*,

Tioga, Palo Alto, CA (1980).

Obe62 A. Oberschelp: *Untersuchungen zur mehrsortigen Quantorenlogik*, Mathematische Annalen 145 (1962).

Ohl87 H.J. Ohlbach: *Link Inheritance in Abstract Clause Graphs*, Journal of Automated Reasoning, Vol. 3, No. 1 (1987), 1-34.

OS86 F. Oppacher, E. Suen: *Controlling deduction with proof condensation and heuristics*, Proc. 8th International Conference on Automated Deduction, Springer LNCS 230 (J.H. Siekmann, Hrsg.), Oxford (1986), 384-393.

OS91 H.J. Ohlbach, J.H. Siekmann: *The Markgraf Karl Refutation Procedure*, in Computational Logic, Essays in Honor of John Alan Robinson, MIT-Press,1991,41-112.

Ric78 M.M. Richter: *Logikkalküle*, Leitfäden der angewandten Mathematik und Mechanik LAMM, Band 43, Teubner, Stuttgart (1978).

Rob65a J.A. Robinson: *A Machine Oriented Logic Based on the Resolution Principle*, J.ACM, Vol. 12, No. 1 (1965), 23-41.

Rob65b J.A. Robinson: *Automated Deduction with Hyper-Resolution*, Intern. Journal of Comp. Mathematics 1 (1965), 227-234.

Sch85 M. Schmidt-Schauß: *A Many-Sorted Calculus with Polymorphic Functions Based on Resolution and Paramodulation*, Proc. of 9th IJCAI, Los Angeles (1985), 1162-1168.

Sho76 R.E. Shostak: *Refutation Graphs*, Artificial Intelligence, Vol. 7, No. 1 (1976), 51-64.

Sho79 R.E. Shostak: *A Graph-Theoretic View of Resolution Theorem-Proving*, Report SRI Internationel, Menlo Park, CA (1979).

Sic76 S. Sickel: *A Search Technique for Clause Interconnectivity Graphs*, IEEE Trans. Comp., Vol. C-25, No. 8 (1976), 823-835.

Sko20 T. Skolem: *Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze*. Skrifter utgit av Videnskapselskapet i Kristiania, Nr. 4 (1920), 4-36.

Smu68 R.M. Smullyan: *First-Order Logic*, Springer Verlag Berlin (1968)

Sti85 M. Stickel: *Automated Deduction by Theory Resolution*, Journal of Automated Reasoning Vol. 1, No. 4 (1985), 333-356.

Tar36 A. Tarski: *Der Wahrheitsbegriff in formalisierten Sprachen*, Studia Philos. 1 (1936).

Wal86 C. Walther: *Automatisches Beweisen*, Kursunterlagen Frühjahrsschule KIFS-86.

Wal87 C. Walther: *A Many-sorted Calculus Based on Resolution and Paramodulation.* Research Notes in Artifical Intelligence, Pitman Ltd., London, M. Kaufmann Inc., Los Altos (1987).

WO84 L. Wos, R. Overbeek, E. Lusk, J. Boyle: *Automated Reasoning – Introduction and*

Applications, Prentice-Hall, Englewood Cliffs, NJ (1984).

Wri89a G. Wrightson: *A Pragmatic Strategy for Clause Graphs or the Strong Completeness of Connection Graphs,* Internal Report 89-3, Dept. of Electrical Engineering and Computer Science, University of Newcastle, Australien, 1989.

Wri89b G. Wrightson: *Clausal Tableaux with Unification*, Internal Report 89-4, Dept. of Electrical Engineering and Computer Science, University of Newcastle, Australien, 1989.