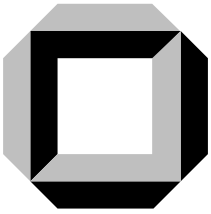
View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by KITopen

Transformation semantischer Tableaus in Klauselmengen mit starren Variablen und in Integer-programming-Probleme

Peter Oel

Interner Bericht 42/96



Universität Karlsruhe
Fakultät für Informatik

Dies ist eine geringfügig überarbeitete Version meiner Diplomarbeit vom 16.4.1996.

Karlsruhe, 20.12.1996

Inhaltsverzeichnis

1 Einleitung

In der Theorie des automatischen Beweisens sind in den letzten Jahrzehnten verschieden Beweiskalküle entwickelt und in automatischen Beweissystemen eingesetzt und ausgebaut worden. Die wohl bekannteste und am weitesten verbreitete Methode ist das Resolutionsverfahren, das 1965 von J. A. Robinson [Rob65] vorgestellt wurde. Neben der Resolutionsmethode hat sich das Tableauverfahren etabliert. Semantische Tableaus wurden von E. W. Beth [Bet86] eingeführt und fanden ihre Verbreitung durch die Darstellung von R. Smullyan [Smu68]. In dieser Diplomarbeit werden semantische Tableaus mit freien Variablen verwendet, die seit Mitte der 80er Jahre untersucht werden [Ree87, Sch87].

Neben der getrennten Weiterentwicklung der Beweisverfahren besteht auch der Wunsch, die einzelnen Methoden zu vergleichen und effiziente Verfeinerungen und Einschränkungen des einen Beweisverfahrens in das andere zu übernehmen. Wegen des unterschiedlichen Aufbaus und der verschiedenen Struktur lassen sich Tableaus und Klauselmengen jedoch nicht direkt ineinander überführen. Der Ablauf eines Beweises im Tableaukalkül unterscheidet sich grundsätzlich von einem Resolutionsbeweis. Deshalb ist es sinnvoll, sich geeignete Verfahren zu überlegen, die Tableaus in Klauselmengen übersetzen und die diese Klauselmengen auch wieder in Tableaus zurückwandeln können.

Einer der Hauptunterschiede von Tableaus gegenüber Klauselmengen ist die Verwendung der Baumstruktur beim Tableauverfahren. Bei der Transformation von Tableaus in Klauselmengen muß darauf geachtet werden, daß diese Struktur in gewisser Weise erhalten bleibt, um einerseits beim Resolutionsverfahren daraus Nutzen zu ziehen und andererseits aus der Klauselmenge wieder ein Tableau herstellen zu können.

In dem dieser Einleitung folgenden Kapitel 2 werden Tableaus mit Verzweigungsvariablen und Klauselmengen mit Verzweigungsvariablen vorgestellt. Die Verwendung von Verzweigungsvariablen zur Erhaltung der Tableaustruktur ist zentraler Bestandteil der in Kapitel 3 vorgestellten Transformationen. Die Transformation Ψ übersetzt ein Tableau mit Verzweigungsliteralen T in eine Klauselmenge mit Verzweigungsliteralen $\Psi(T)$. Die zweite Transformation Φ erzeugt zu einer Klauselmenge K Tableaus $\Phi(K)$. Diese Transformation ist, wie in diesem Kapitel ge-

zeigt wird, nicht immer eindeutig. Dieses Problem kann mit der Verwendung einer Ordnung auf den Verzweigungsliteralen umgangen werden. Die Bildung eines Tableaus aus einer Klauselmenge ist insgesamt aufwendiger als die Transformation in umgekehrter Richtung, da die Tableaustruktur wieder hergestellt werden muß.

Im darauffolgenden Kapitel 4 werden die Eigenschaften der Transformationen untersucht. Es wird gezeigt, daß Φ und Ψ korrekt sind. Außerdem wird gezeigt, daß die Anwendung von Φ auf das Ergebnis der Transformation Ψ wieder das Ausgangstableau ergibt, also $\Phi(\Psi(T)) = T$. Der umgekehrte Fall ist schwieriger, aber unter bestimmten Voraussetzungen gilt auch $\Psi(\Phi(K)) = K$ für eine Klauselmenge K.

Die Erfüllbarkeit einer Klauselmenge kann mit Integer-programming-Methoden getestet werden. Ein Integer-Programm [SM89, Hoo93] ist ein mathematisches Modell, bei dem eine Menge von positiven, ganzzahligen Werten gesucht wird, die eine Gleichung maximieren, während sie ein System von Bedingungen, sogenannten Integer-programming-constraints, erfüllen. Sind nur Werte aus der Menge $\{0, 1\}$ zugelassen, so spricht man von 0-1-Integer-Programmen. In Kapitel 5 wird eine einfache Übersetzung von Klauselmengen mit Verzweigungsliteralen in 0-1-Integer-Programme angegeben und ein Verfahren vorgestellt, mit dem die Unerfüllbarkeit überprüft werden kann.

Nachdem die Transformationen eingeführt sind, ist es möglich, in Kapitel 6 zu untersuchen, inwiefern bestimmte Verfeinerungen und Einschränkungen von einem Beweisverfahren in das andere übertragen werden können.

Schließlich wird in Kapitel 7 eine Programmierung der Transformationen Φ und Ψ angegeben.

2 Klauselmengen und Tableaus

Dieses Kapitel beschreibt die grundlegenden Definitionen für Klauselmengen und Tableaus. Nach einigen allgemeinen Festlegungen werden in den Abschnitten 2.2 und 2.3 Verzweigungsliterale und Ordnungen auf ihnen eingeführt. Verzweigungsliterale sind von elementarer Bedeutung für die Transformationen zwischen Klauselmengen und Tableaus, da mit ihnen unter Zuhilfenahme einer Ordnung der Zusammenhang zwischen einer Klauselmenge und einem bestimmten Tableau hergestellt werden kann. In Abschnitt 2.4 werden Tableaus mit Verzweigungsliteralen und ein Tableauverfahren auf ihnen vorgestellt. Schließlich werden in Abschnitt 2.5 noch Klauselmengen mit Verzweigungsliteralen und ein Resolutionsverfahren definiert.

2.1 Allgemeine Definitionen

Die in dieser Arbeit verwendeten logischen Begriffe werden, falls nicht anders angegeben, in herkömmlicher Weise verwendet, wie zum Beispiel in [Fit90] definiert. Alle Aussagen sind in Prädikatenlogik erster Ordnung formuliert.

Häufig werden folgende Schreibweisen verwendet: Das zu l komplementäre Literal wird mit $\overline{l}$ angegeben. Für die Menge von Literalen $K = \{l_1, \ldots, l_n\}$ ist $\overline{K}$ definiert durch $\overline{K} = \{\overline{l_1}, \ldots, \overline{l_n}\}$. Mit $\rho(l)$ erhält man die zum Literal l gehörende atomare Formel, zum Beispiel: $\rho(\neg p(f(x))) = \rho(p(f(x))) = p(f(x))$. Die Menge der in C frei vorkommenen Variablen heißt $var(C)$. Außerdem gilt für die Klauselmenge $M = \{C_1, \ldots, C_n\}$ und die Substitution σ, daß $M\sigma = \{C_1\sigma, \ldots, C_n\sigma\}$.

2.2 Verzweigungsliterale

Für die Transformation von Tableaus in Klauselmengen und vor allem für die Rücktransformation müssen die Äste im Tableau genauer gekennzeichnet werden, damit es möglich ist, Formeln an der richtigen Stelle in das Tableau einzufügen. Dies geschieht mit Verzweigungsliteralen. Diese Verzweigungsliterale werden, wie der Name schon vermuten läßt, mit den Verzweigungen im Tableau verbunden. Damit kann man die Pfade im Tableau eindeutig benennen.

2.3 Verzweigungsordnungen

Ordnungen werden bei der Transformation von Klauselmengen in Tableaus verwendet. Meistens ist es möglich, aus einer Klauselmenge mehrere verschiedene Tableaus zu erzeugen. Um ein bestimmtes dieser Tableaus zu erhalten, muß die Reihenfolge der im Tableau auftretenden Verzweigungen festgelegt werden. Dazu wird eine Ordnungsrelation, im folgenden auch einfach *Ordnung* genannt, verwendet, die die Verzweigungsliterale ordnet. Diese Ordnungsrelation wird jedoch nicht auf den Verzweigungsliteralen selbst, sondern auf der Menge der atomaren Formeln aller Verzweigungsliterale definiert.

Definition 2.1 Ordnungsrelation, Ordnung
Sei R eine binäre Relation auf der Menge M.

- *R heißt strikte, partielle Ordnungsrelation (oder Ordnung) auf M, falls R transitiv und irreflexiv ist.*
- *R heißt totale Ordnungsrelation (oder Ordnung) auf M, falls R eine strikte, partielle Ordnung auf M ist und für alle $m_1 \in M$ und $m_2 \in M$ mit $m_1 \neq m_2$ gilt: Entweder $m_1 R m_2$ oder $m_2 R m_1$.*

Definition 2.2 Verzweigungsordnung
Sei M eine Menge von Verzweigungsliteralen und At_M die Menge aller atomaren Formeln von Elementen aus M. $>_M$ heißt Verzweigungsordnung(srelation), wenn $>_M$ eine strikte, partielle Ordnung in der Menge At_M ist.

Es kann auch die Schreibweise $>_K$ oder $>_T$ verwendet werden, wobei K eine Klauselmenge und T ein Tableau ist. $>_K$ und $>_T$ sind dann Verzweigungsordnungen auf den atomaren Formeln aller Verzweigungsliterale in K und T.

Für die Verzweigungsatome $p \in M$ und $q \in M$ gilt $p \not>_M q$, falls $q >_M p$ oder falls p und q bezüglich $>_M$ nicht vergleichbar sind. Die Verzweigungsatome p und q heißen unvergleichbar bezüglich $>_M$ ($p <>_M q$), wenn weder $p >_M q$ noch $q >_M p$ gilt.

In einer Menge von Verzweigungsatomen kann es bezüglich einer Ordnungsrelation maximale Elemente geben. Diese sind wie folgt definiert:

Definition 2.3 Maximales Element
Sei M eine Menge von Verzweigungsliteralen und $>_M$ eine Verzweigungsordnung in M. Dann heißt $p \in M$ maximales Element von M bezüglich $>_M$, falls $\forall q, q \in M : \rho(q) \not>_M \rho(p)$.

Falls $>_M$ eine partielle Ordnung auf M ist, kann es mehrere maximale Elemente geben. Ist $>_M$ total, dann gibt es ein eindeutiges maximales Element.

Definition 2.4 $max_>$
Sei M eine Menge von Verzweigungsliteralen und $>_M$ eine Verzweigungsordnung. Dann ist $max_{>_M}(M) = \{p_1, \ldots, p_n\}$ die Menge aller maximalen Elemente von M, falls $\forall\, p_i \in M$, $1 \leq i \leq n$: p_i ist maximales Element von M bezüglich $>_M$.

In gleicher Weise werden minimale Elemente definiert:

Definition 2.5 Minimales Element
Sei M eine Menge von Verzweigungsliteralen und $>_M$ eine Verzweigungsordnung in M. Dann heißt $p \in M$ minimales Element von M bezüglich $>_M$, falls $\forall q, q \in M : \rho(p) \not>_M \rho(q)$.

Definition 2.6 $min_>$
Sei M eine Menge von Verzweigungsliteralen und $>_M$ eine Verzweigungsordnung. Dann ist $min_{>_M}(M) = \{p_1, \ldots, p_n\}$ die Menge aller minimalen Elemente von M, falls $\forall\, p_i \in M$, $1 \leq i \leq n$: p_i ist minimales Element von M bezüglich $>_M$.

Zur Berechnung der maximalen Verzweigungsatome einer Klausel C kann $max_>$ auch direkt auf C angewendet werden. Dann ist $max_{>_K}(C) = max_{>_K}(M)$, wobei $M \subseteq C$ die Teilmenge von C ist, die genau alle Verzweigungsliterale aus C enthält. In gleicher Weise wird $min_{>_K}(C)$ verwendet.

2.4 Tableaus

Als Tableaus werden Tableaus mit freien Variablen verwendet, wie zum Beispiel in [Fit90]. Es wird hier jedoch eine Einschränkung vorgenommen. Auf den Ästen dürfen nur Disjunktionen von Literalen vorkommen.

Definition 2.7 Tableauformel
A ist eine Tableauformel, wenn A eine Disjunktion von Literalen ist und in A entweder alle Variablen frei vorkommen, oder alle Variablen durch Allquantoren gebunden sind.

Es handelt sich also eigentlich um Klauseltableaus. Im folgenden wird jedoch nicht von Klauseltableaus gesprochen, sondern die Tableaus werden wie semantische Tableaus mit freien Variablen eingeführt. Diese Einschränkung der im Tableau zugelassenen Formeln vereinfacht die Transformation zwischen Tableaus und Klauselmengen, sie ist aber nicht zwingend. Ohne sie werden die Transformationsschritte komplexer.

Bei der Disjunktion von Literalen wird keine explizite Klammerung angegeben. Die später eingeführte β-Tableauregelanwendung ist jedoch zweistellig definiert. Um den korrekten Ablauf des Tableauverfahrens zu gewährleisten, wird für die Disjunktionen eine Klammerung implizit in sinnvoller Weise angenommen.

2.4.1 Tableauverfahren mit Verzweigungsliteralen

Beim Aufbau der Tableaus wird die *uniforme Notation* [Fit90] verwendet, das heißt, die Menge der Formeln wird in vier Klassen eingeteilt: α für konjunktive Formeln, β für disjunktive Formeln, γ für universell quantifizierte Formeln und δ für existentiell quantifizierte Formeln. Außerdem werden die bekannten Tableauerweiterungsregeln verwendet. Durch die Verwendung der Tableauformeln (Definition 2.7) werden nur die β- und die γ-Erweiterungsregel benötigt. Die Tableauerweiterungsregeln und die verschiedenen Klassen von Formeln sind in den Abbildungen 2.1 und 2.2 zu sehen.

$$\frac{\beta}{\begin{array}{c|c}\beta_1 & \beta_2\end{array}} \qquad j(x_1, \ldots, x_n)$$

Wobei j ein neues Verzweigunsprädikat ist und $x_1, \ldots, x_n$ die sowohl in β_1 als auch in β_2 frei vorkommenden Variablen sind.

$$\frac{\gamma}{\gamma_1(x_1, \ldots, x_n)}$$

Wobei $x_1, \ldots, x_n$ neue freie Variablen sind.

Abbildung 2.1: Tableauerweiterungsregeln

Bei der Erweiterung eines Tableaus durch Anwendung einer β-Regel wird die Verzweigung mit einem Verzweigungsprädikat $j(x_1, \ldots, x_n)$ gekennzeichnet, wobei j ein neues Prädikatensymbol ist und $x_1, \ldots, x_n$ die in β_1 und β_2 frei vorkommenden Variablen (bzw. deren Schnittmenge) sind. Die beiden neu entstandenen Äste im Tableau werden durch die Verwendung der positiven $j(x_1, \ldots, x_n)$ und der negativen $\neg j(x_1, \ldots, x_n)$ Ausprägung des neuen Verzweigungsprädikats gekennzeichnet.

Ein Tableau wird als endlicher binärer Baum dargestellt, dessen Knoten prädikatenlogische Formeln sind. Der Baum wird wie folgt konstruiert:

Definition 2.8 Tableau
Sei $F = \{f_1, \ldots, f_n\}$ eine endliche Menge von Tableauformeln. Dann ist der

lineare Baum

$$f_1$$
$$\vdots$$
$$f_n$$

ein Tableau mit Verzweigungsliteralen für F.
Ist T ein Tableau mit Verzweigungsliteralen für F und folgt T' aus T durch Anwendung einer Tableauerweiterungsregel aus Bild 2.1, dann ist T' ein Tableau mit Verzweigungsliteralen für F.

α	α_1	α_2
$\phi \wedge \psi$	ϕ	ψ
$\neg(\phi \vee \psi)$	$\neg\phi$	$\neg\psi$
$\neg(\phi \supset \psi)$	ϕ	$\neg\psi$
$\neg\neg\phi$	ϕ	ϕ

β	β_1	β_2
$\phi \vee \psi$	ϕ	ψ
$\neg(\phi \wedge \psi)$	$\neg\phi$	$\neg\psi$
$\phi \supset \psi$	$\neg\phi$	ψ

γ	$\gamma_1(x)$
$\forall z\phi(z)$	$\phi(x)$
$\neg\exists\phi(z)$	$\neg\phi(x)$

δ	$\delta_1(f(x_1,\ldots,x_n))$
$\neg\forall z\phi(z)$	$\neg\phi(f(x_1,\ldots,x_n))$
$\exists\phi(z)$	$\phi(f(x_1,\ldots,x_n))$

Abbildung 2.2: Die verschiedenen Formeltypen.

2.4.2 Notation für Tableaus mit Verzweigungsliteralen

Für die Transformationen wird eine kompaktere Notation benötigt. Wegen der einfacheren Beschreibung werden lineare Teilbäume in einem Tableau als ein Multiknoten betrachtet. Ein solcher Knoten kann mehrere Formeln enthalten. Da die Reihenfolge der Formeln in einem linearen Teiltableau beliebig ist, kann ein Multiknoten als Menge von Tableauformeln betrachtet werden. Die γ-Regelanwendungen finden innerhalb eines Multiknotens statt.

Außerdem wird ein Tableau als ein Paar, bestehend aus einem Multiknoten und Unterbäumen, dargestellt. Hier werden immer zwei Unterbäume verwendet. $T = \langle W, ((j(x_1,\ldots,x_n), T_1), (\neg j(x_1,\ldots,x_n), T_2))\rangle$ ist also ein Tableau, wobei W der Multiknoten ist und T_1 und T_2 die beiden Unterbäume von T sind. Die Unterbäume sind mit dem Verzweigungsprädikat j beziehungsweise dessen positiver und negativer Ausprägung gekennzeichnet.

Die Verwendung von Multiknoten und das Tableaus T sind in Abbildung 2.3 zu sehen.

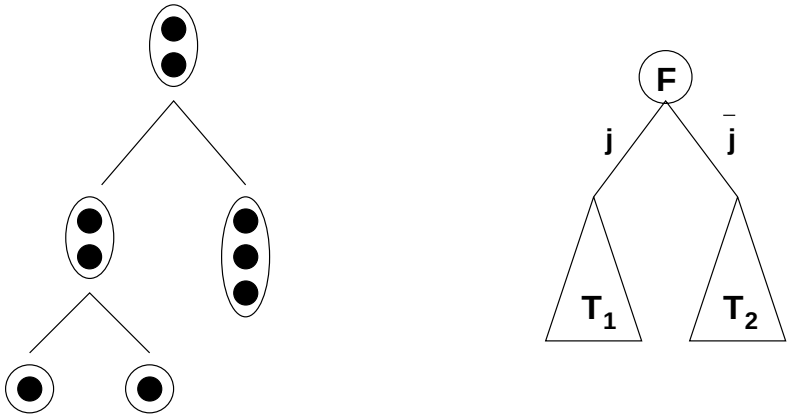


Abbildung 2.3: Tableau mit Multiknoten und das Tableau $T = \langle F, (j, T_1), (\bar{j}, T_2)\rangle$.

Mit dieser Notation ist ein Tableau mit Verzweigungsliteralen auf folgende Weise definiert:

Definition 2.9 Tableau mit Verzweigungsliteralen
Sei $F = \{f_1, \ldots, f_n\}$ eine endliche Menge von Tableauformeln.

1. *Dann ist $T = \langle \{f_1, \ldots, f_n\}, \emptyset\rangle$ ein Tableau mit Verzweigungsliteralen für F.*
2. *Falls $T = \langle W, U\rangle$ ein Tableau mit Verzweigungsliteralen für F ist, dann ist auch T' ein Tableau mit Verzweigungsliteralen für F, wobei T' durch Ersetzen eines Knotens $S = \langle W_1, U_1\rangle$ mit $\gamma \in W_1$ aus T durch $S' = \langle W_1 \cup \{\gamma_1\}, U_1\rangle$ entsteht.*
3. *Falls $T = \langle W, U\rangle$ ein Tableau mit Verzweigungsliteralen für F ist, dann ist auch T' ein Tableau mit Verzweigungsliteralen für F. Dabei ist β Element eines Multiknotens M in T, und T' entsteht durch Ersetzen aller Blätter $B_i = \langle W_i, \emptyset\rangle$ im Teilbaum mit der Wurzel M durch $B_i' = \langle W_i, ((j, \langle\{\beta_1\}, \emptyset\rangle), (\bar{j}, \langle\{\beta_2\}, \emptyset\rangle))\rangle$.*
4. *Falls $T = \langle W, U\rangle$ ein Tableau mit Verzweigungsliteralen für F ist und $S = \langle W_1, U_1\rangle$ ein Knoten in T mit dem Literal $f \in W_1$, und falls es eine Substitution σ und ein Literal f' auf dem Pfad von der Wurzel nach S gibt, mit $f'\sigma = \overline{f}\sigma$, dann ist auch T' ein Tableau mit Verzweigungsliteralen für F. Dabei entsteht T' aus T durch Ersetzen von S durch $S' = \langle W_1 \cup \{\bot\}, U_1\rangle$ und Anwendung der Substitution σ auf T.*

Dabei sind γ, γ_1, β, β_1, β_2 und j entsprechend der Tableauerweiterungsregeln aus Bild 2.1 definiert.

Bei einer β-Regel-Erweiterung wird, wie in Fall 3 der Definition 2.9 beschrieben, die neue Verzweigung an alle Blätter des Teilbaums, in dem die β-Formel an der Wurzel steht, angehängt. Das führt nicht unbedingt zu einem effizienten Tableauverfahren, da diese β-Expansionen meistens nicht in allen Teilbäumen benötigt werden. Es kann statt dessen auch folgende Erweiterung verwendet werden:

Falls $T = \langle W, U\rangle$ ein Tableau mit Verzweigungsliteralen für F ist und $B = \langle W_1, \emptyset\rangle$ ein Blatt in T, dann ist auch T' ein Tableau mit Verzweigungsliteralen für F. Dabei ist β Element eines Multiknotens auf dem Pfad von der Wurzel nach B, und T' entsteht durch Ersetzen von B durch $B' = \langle W_1, ((j, \langle\{\beta_1\}, \emptyset\rangle), (\overline{j}, \langle\{\beta_2\}, \emptyset\rangle))\rangle$.

Bei der Anwendung einer Substitution σ auf ein Tableau mit Verzweigungsliteralen T wird σ auf alle Multiknoten und alle Verzweigungsliterale in T angewendet.

Ein *Ast* A im Tableau T ist ein maximaler Pfad von der Wurzel zu einem Blatt in T. Er läßt sich durch eine Menge von Verzweigungsliteralen eindeutig angeben. Ein Ast im Tableau mit Verzweigungsliteralen heißt *geschlossen*, wenn in einem Multiknoten auf dem Ast ein Abschlußsymbol $\perp$ vorkommt, das heißt, wenn er zwei komplementäre Literale enthält. Ein Tableau T mit Verzweigungsliteralen heißt geschlossen, wenn alle Äste in T geschlossen sind. Für den späteren Vergleich des Tableau- und des Resolutionsverfahrens ist es sinnvoll, mit dem Abschlußsymbol $\perp$ auch die Substitution σ zu speichern, die zu dem Abschluß geführt hat.

2.4.3 Aussagen über Tableaus mit Verzweigungsliteralen

Die Verzweigungsliterale der im letzten Abschnitt eingeführten Tableaus werden nur als Bezeichner verwendet und haben keinen Einfluß auf das Tableauverfahren. Das hier verwendete Abschlußsymbol $\perp$ wird in [Fit90] als Konstante mit dem Wahrheitswert *falsch* verwendet. Das stellt jedoch keinen Unterschied dar, da ein Ast, der die Konstante $\perp$ enthält, automatisch geschlossen ist.

Es gelten für Tableaus mit Verzweigungsliteralen also die in [Fit90] bewiesenen Sätze:

Satz 2.10
Sei F eine Menge von Tableauformeln. Gibt es ein geschlossenes Tableau mit Verzweigungsliteralen für F, dann ist F unerfüllbar.

Satz 2.11
Sei F eine Menge von Tableauformeln. Ist F unerfüllbar, dann gibt es ein geschlossenes Tableau mit Verzweigungsliteralen für F.

2.5 Klauselmengen

Die bei den Transformationen verwendeten Klauselmengen unterscheiden sich von den "normalen" Klauselmengen, wie sie zum Beispiel in [Lei92, CL73] für Resolutionsverfahren angegeben sind. Die hier verwendeten Klauselmengen sollen *Klauselmengen mit Verzweigungsliteralen* heißen. Neben der Verwendung von Verzweigungsliteralen wird die Klauselmenge noch in eine starre und eine universelle Menge unterteilt. Diese Unterteilung ist für das Resolutionsverfahren auf Klauselmengen mit Verzweigungsliteralen wichtig.

2.5.1 Klauseln mit Verzweigungsliteralen

Wurden die Verzweigungsliterale im Tableau nur zur Namensgebung verwendet, so werden sie in den Klauseln wie "normale" Literale behandelt. Eine Klausel mit Verzweigungsliteralen kann also neben diesen "normalen" Literalen, die im folgenden *Formelliterale* genannt werden, auch Verzweigungsliterale enthalten. Das ist jedoch nicht zwingend. Insbesondere ist jede "normale" Klausel auch eine Klausel mit Verzweigungsliteralen. Außerdem ist es möglich, daß eine Klausel mit Verzeigungsliteralen keine Formelliterale enthält.

Um die Verzweigungsliterale und die Formelliterale einer Klausel C zu erhalten, werden die Funktionen Vl und Fl verwendet, die auf folgende Weise definiert sind:

Definition 2.12 Vl und Fl
Sei $C = \{j_1, \ldots, j_n, l_1, \ldots, l_m\}$ eine Klausel mit den Verzweigungsliteralen $j_1, \ldots, j_n$ und den Formelliteralen $l_1, \ldots, l_m$. Dann ist $Vl(C) = \{j_1, \ldots, j_n\}$ die Menge aller Verzweigungsliterale in C, und $Fl(C) = \{l_1, \ldots, l_m\}$ ist die Menge aller Formelliterale in C.

Außerdem sind Klauseln mit Verzweigungsliteralen Multimengen, das heißt, das gleiche Literal kann mehrmals in einer Klausel vorkommen. Das erfordert beim Resolutionsverfahren einen expliziten Faktorisierungsschritt.

2.5.2 Klauselmengen mit Verzweigungsliteralen

Klauselmengen mit Verzweigungsliteralen sind Mengen von Klauseln mit Verzweigungsliteralen. Bei der Anwendung eines Resolutionsverfahrens auf eine Klauselmenge mit Verzweigungsliteralen werden die Verzweigungsliterale genauso behandelt, wie die Formelliterale. Syntaktisch besteht also kein Unterschied zwischen Verzweigungsliteralen und Formelliteralen.

In einer einzelnen Klausel können beliebig viele Verzweigungsliterale vorkommen. In einer Klauselmenge ist das Vorkommen von Verzweigungsliteralen in den einzelnen Klauseln jedoch nicht beliebig. Die Verzweigungsliterale geben den Aufbau der Formel wieder, die die Klauselmenge repräsentiert. Eine allgemeine Definition für Klauselmengen mit Verzweigungsliteralen wird erst im nächsten Abschnitt gegeben, da dafür die Definitionen weiterer Begriffe notwendig sind.

2.5.3 Erzeugen von Klauselmengen mit Verzweigungsvariablen

Hier sollen zunächst zwei Verfahren beschrieben werden, die aus einer beliebigen prädikatenlogischen Formel F eine Klauselmenge mit Verzweigungsliteralen erzeugen:

Das erste ist eine triviale Transformation, da keine Verzweigungsliterale erzeugt werden. Jede "normale" Klauselmenge für die Formel F, die zum Beispiel durch Skolemisieren und Transformation in konjunktive Normalform erzeugt wurde (wie zum Beispiel in [Sch89] beschrieben), ist auch eine Klauselmenge mit Verzweigungsliteralen für F.

Das zweite Verfahren zur Erzeugung einer Klauselmenge mit Verzweigungsliteralen für eine Formel F ist eine Abwandlung eines in [BH96] vorgestellten Algorithmus zur Transformation von F in ein Integer-programming-Problem. Das Verfahren kann fast analog für Klauselmengen verwendet werden. Anstelle der Integer-programming-constraints werden Klauseln mit Verzweigungsliteralen erzeugt, und anstelle von neuen Interger-programming-Variablen werden neue Verzweigungsliterale eingeführt.

Bei diesem Verfahren werden markierte Formeln $\boxed{C}\ F$ verwendet, wobei C eine Klausel und F eine Formel ist. Damit ergibt sich folgender Ablauf:

Sei F eine beliebige prädikatenlogische Formel und $M_0, \ldots, M_n$ eine Folge von Mengen, für die die folgenden drei Bedingungen gelten:

- $M_0 = \{\boxed{\emptyset}\ F\}$,
- M_k entsteht aus M_{k-1} durch Ersetzen von $g \in M_{k-1}$ durch g', wobei durch Anwendung einer Erweiterungsregel aus Bild 2.4 auf g entsteht, (Anstelle der β-Regel kann, falls möglich, auch eine der optimierten β-Regeln verwendet werden.)
- M_n enthält nur noch Klauseln.

M_n ist dann eine Klauselmenge mit Verzweigungsliteralen für die Formel F.

Beide Verfahren lassen sich in gleicher Weise auch auf Formelmengen anwenden.

Falls beim zweiten Verfahren in keinem Schritt eine optimierte β-Regel verwendet wird, erhält man eine Klauselmenge K, in der für jede Klausel $C \in K$ gilt: $|Fl(C)| = 1$. Jede Klausel enthält also nur ein Formelliteral. Wird das zweite Verfahren auf eine Menge von Tableauformeln (Definition 2.7) angewendet, so ist es möglich, neben γ-Regeln nur optimierte β-Regeln zu verwenden. Die resultierende Klauselmenge enthält dann keine Verzweigungsliterale.

Optimierte β-Regeln:

$$\frac{\boxed{C}\ \beta}{\begin{array}{l}\boxed{C \cup \{j(x_1,\ldots,x_n)\}}\ \beta_1 \\ \boxed{C \cup \{\neg j(x_1,\ldots,x_n)\}}\ \beta_2\end{array}} \qquad \frac{\boxed{C}\ \beta}{\boxed{C \cup \{\beta_1\}}\ \beta_2} \qquad \frac{\boxed{C}\ \beta}{\boxed{C \cup \{\beta_2\}}\ \beta_1}$$

Wobei j ein neues Verzweigungsprädikatsymbol ist und $\{x_1,\ldots,x_n\}$ die Schnittmenge der freien Variablen aus β_1 und β_1.

Falls $\beta_1 = p$ und p Literal.

Falls $\beta_2 = p$ und p Literal.

$$\frac{\boxed{C}\ \alpha}{\begin{array}{l}\boxed{C}\ \alpha_1 \\ \boxed{C}\ \alpha_2\end{array}} \qquad \frac{\boxed{C}\ p}{C \cup \{p\}} \qquad \frac{\boxed{C}\ \gamma}{\boxed{C}\ \gamma_1(x)} \qquad \frac{\boxed{C}\ \delta}{\boxed{C}\ \delta_1(f(x_1,\ldots,x_n))}$$

p Literal

Wobei x eine neue freie Variable ist.

Wobei f ein neues Skolemfunktionssymbol ist und $x_1,\ldots,x_n$ die in δ frei vorkommenden Variablen sind.

Abbildung 2.4: Erweiterungsregeln zur Erzeugung von Klauselmengen.

2.5.4 Einteilung in starre und universelle Teilmengen

Eine für das im nächsten Abschnitt beschriebene Resolutionsverfahren benötigte Klauselmenge K besteht aus zwei unterschiedlich verwendeten Teilmengen, der *starren Teilmenge* und der *universellen Teilmenge*. Diese seien mit ST_K und UT_K bezeichnet und $K = \langle ST_K, UT_K \rangle$.

Die Einteilung der Klauseln in eine starre und eine universelle Teilmenge funktioniert für die beiden im vorherigen Abschnitt beschriebenen Verfahren auf die

gleiche Weise:

Die Klauseln einer Klauselmenge $K = \langle ST_K, UT_K \rangle$ werden in die starre und die universelle Teilmenge so eingeordnet, daß für alle $C \in K$ gilt: Falls $var(C) = \emptyset$, dann ist $C \in ST_K$, sonst $C \in UT_K$.

Diese Einordnungsregel gilt jedoch nur für die beiden oben vorgestellten Verfahren. Es kann im Verlauf einer Resolutionsableitung auch Klauselmengen mit Verzweigungsliteralen geben, bei denen Klauseln aus der starren Teilmenge freie Variablen enthalten.

2.5.5 Resolutionsverfahren

Resolutionverfahren, wie sie zum Beispiel in [CL73, Lei92] vorgestellt werden, verwenden in der Regel nur eine universelle Klauselmenge. Das heißt, eine Klausel entspricht einer all-quantifizierten Disjunktion von Literalen. Damit sind die einzelnen Klauseln variablendisjunkt und die bei Unifikationen durchzuführenden Substitutionen sind nur für die Ergebnisklausel relevant.

Im Hinblick auf die später eingeführten Transformationen zwischen Tableaus und Klauselmengen ist ein Resolutionsverfahren, das nur universelle Klauselmengen verwendet, nicht geeignet. Aus diesem Grund wird hier neben der universellen Teilmenge noch eine starre Teilmenge verwendet. Die in dieser Teilmenge vorkommenden Klauseln müssen nicht variablendisjunkt sein. Die bei Resolventen- und Faktorbildung erhaltenen Substitutionen müssen dann jedoch auf die gesamte Klauselmenge angewendet werden.

Dieser Ansatz besitzt viele Freiheisgrade und wird daher ohne größere Einschränkungen nicht als effizientes Resolutionsverfahren geeignet sein. Insbesondere ist zu klären, in welcher der Teilmengen Resolventen gebildet werden dürfen und unter welchen Bedingungen Klauseln aus der einen in die andere Teilmenge überführt werden können.

Im hier vorgestellten Verfahren ist die starre Teilmenge die eigentliche Klauselmenge. Resolventen- und Faktorbildungen sind nur in ihr erlaubt. Die universelle Teilmenge wird nur zum Erzeugen weiterer Instanzen der Ursprungsklauseln verwendet, die dann in die starre Teilmenge überführt werden.

Zunächst werden jetzt Resolventen und Faktoren definiert, außerdem noch Verzweigungsklauseln.

Definition 2.13 Resolvente

Sei K eine (starre) Klauselmenge mit $C_1 \in K$ und $C_2 \in K$ und $l_1 \in C_1$ und $l_2 \in C_2$. Außerdem sei $\{\bar{l}_1, l_2\}$ unifizierbar durch den allgemeinsten Unifikator σ. Dann heißt $R = (C_1 - \{l_1\})\sigma \cup (C_2 - \{l_2\})\sigma$ Resolvente von C_1 und C_2.

Definition 2.14 Faktor
Sei C eine Klausel und $A \subseteq C$ eine Teilmenge von C mit $A \neq \emptyset$. Ist A unifizierbar durch einen allgemeinsten Unifikator σ, dann wird $C\sigma$ Faktor von C genannt.

Definition 2.15 Verzweigungsklauseln
Sei K eine Klauselmenge und $C = \{j_1, \ldots, j_n, l_1, \ldots, l_m\}$ eine Klausel aus K. Dann heißen die Klauseln $D = \{j_1, \ldots, j_n, j_{n+1}, l_1\}$ und $E = \{j_1, \ldots, j_n, \neg j_{n+1}, l_2, \ldots, l_m\}$ Verzweigungsklauseln von C, wobei j_{n+1} ein neues Prädikatensymbol ist, das bisher in keiner Klausel vorkommt. Kommen in den Literalen $l_1, \ldots, l_m$ freie Variablen vor, so ist das neue Verzweigungsliteral k-stellig: $j_{n+1}(x_1, \ldots, x_k)$. Dabei ist $\{x_1, \ldots, x_k\}$ die Schnittmenge der in l_1 und $l_2, \ldots, l_m$ frei vorkommenden Variablen.

Die Verwendung einer universellen und einer starren Teilmenge bei der Klauselmenge erfordert eine genauere Festlegung, in welcher der Teilmengen Resolventen, Faktoren und Verzweigungsklauseln gebildet werden dürfen. In der Definition für erweiterte Klauselmengen wird festgelegt, daß Erweiterungen nur in der starren Teilmenge vorgenommen werden dürfen. Zusätzlich ist noch die Überführung von Klauseln aus der universellen in die starre Teilmenge erlaubt.

Definition 2.16 Erweiterte Klauselmenge
Sei $K = \langle ST_K, UT_K \rangle$ eine Klauselmenge mit Verzweigungsliteralen und seien ST_K und UT_K deren starre und universelle Teilmengen. $K' = \langle ST_{K'}, UT_{K'} \rangle$ ist eine erweiterte Klauselmenge von K, falls K' aus K entsteht durch

1. *Hinzufügen einer neuen Klausel D, wobei D eine Resolvente zweier Klauseln aus ST_K oder ein Faktor einer Klausel aus ST_K ist und die Substitution σ auch auf ST_K angewendet wird ($K' = \langle ST_K\sigma \cup \{D\}, UT_K \rangle$), oder durch*

2. *Hinzufügen einer neuen Klausel D, wobei D durch Anwendung einer Variablenumbenennung μ auf eine Klausel E aus der universellen Teilmenge UT_K entsteht ($E \in UT_K$, $D = E\mu$, $K' = \langle ST_K \cup \{D\}, UT_K \rangle$), oder durch*

3. *Hinzufügen zweier neuer Klauseln D und E, wobei D und E die Verzweigungsklauseln einer Klausel aus ST_K sind ($K' = \langle ST_K \cup \{D, E\}, UT_K \rangle$).*

Damit ist es möglich, Klauselmengen mit Verzweigungsliteralen und Resolutionsableitungen genauer einzuführen.

Definition 2.17 Klauselmenge mit Verzweigungsliteralen
K ist eine Klauselmenge mit Verzweigungsliteralen für die Formel F, falls

1. *K mit Hilfe des trivialen Verfahrens aus Abschnitt 2.5.3 aus F entstanden ist, oder*
2. *K durch das zweite in Abschnitt 2.5.3 beschriebenen Verfahren aus der Formel F erzeugt wurde, oder*
3. *K eine erweiterte Klauselmenge von K' ist, wobei K' eine Klauselmenge mit Verzweigungsliteralen für F ist.*

Definition 2.18 Resolutionsableitung mit starren Variablen
Eine Resolutionsableitung der Klausel C aus einer Klauselmenge mit Verzweigungsliteralen K_1 ist eine Folge von Klauselmengen $K_1, \ldots, K_n$. Dabei ist $C \in K_n$, und für alle i mit $1 < i \leq n$ gilt, daß K_i eine erweiterte Klauselmenge von K_{i-1} ist.

2.5.6 Aussagen über Klauselmengen mit Verzweigungsliteralen

Die Verzweigungsliterale werden beim Resolutionsverfahren auf Klauselmengen mit Verzweigungsliteralen wie die "normalen" Literale verwendet. Die Verwendung von Verzweigungsklauseln beeinflußt Aussagen über die Erfüllbarkeit nicht. Ist die Klauselmenge K erfüllbar, dann ist auch die um Verzweigungklauseln erweiterte Klauselmenge K' erfüllbar.

Für Klauselmengen mit Verzweigungsliteralen gelten also auch die in [Häh94a] bewiesenen Sätze:

Satz 2.19
Sei K eine Klauselmenge mit Verzweigungsliteralen. Wenn es eine Resolutionsableitung der leeren Klausel aus K gibt, dann ist K unerfüllbar.

3 Transformationen

Nachdem im letzten Kapitel Tableaus und Klauselmengen mit Verzweigungsliteralen eingeführt wurden, werden in diesem die Transformationen zwischen diesen Klauselmengen und Tableaus erklärt. Klauseln, Klauselmengen und Tableaus sind im folgenden immer Klauseln, Klauselmengen und Tableaus mit Verzweigungsliteralen, so daß die explizite Angabe von "mit Verzweigungsliteralen" nicht immer verwendet werden muß. Zunächst (Abschnitt 3.2) wird noch einmal auf die Rolle der Verzweigungsliterale eingegangen. In Abschnitt 3.3 wird dann die Transformation von Klauselmengen in Tableaus definiert und im folgenden Abschnitt 3.4 das Gegenstück, die Transformation von Tableaus in Klauselmengen. Jeweils am Ende dieser beiden Abschnitte wird ein umfangreiches Beispiel zur Erläuterung der Transformationen gegeben.

3.1 Wertebereiche der Funktionen

Um die Wertebereiche der Transformationsfunktionen besser angeben zu können, werden Bezeichner verwendet. Die zugeordneten Symbole sind in der Tabelle in Abbildung 3.1 zu sehen.

Verzweigungsordnung	$\mathcal{O}$
Tableauformel	$\mathcal{F}$
Multiknoten	$\mathcal{W}$
Literal	$\mathcal{L}$
Klausel	$\mathcal{C}$
Klauselmenge	$\mathcal{K}$
Tableau	$\mathcal{T}$
Menge von Tableaus	$\mathcal{T}^*$

Abbildung 3.1: Die Symbole der verwendeten Funktionswertebereiche.

3.2 Semantik der Verzweigungsliterale

Im Tableau kommt den Verzweigungsliteralen nur eine bezeichnende Rolle zu. Eine Verzweigung wird mit einem Verzweigungsatom gekennzeichnet beziehungsweise die beiden Teilbäume jeweils mit der positiven und der negative Ausprägung des Literals. Für diese Markierung ist es nicht notwendig, Literale zu verwenden. Die Verzweigungen könnten auch einfach durchnummeriert werden. Die Bezeichner sollen jedoch den Bezug zwischen Tableaus und Klauselmengen herstellen. Da sie in den Klauselmengen als "normale" Literale verwendet werden, ist es sinnvoll, auch schon im Tableau Literale als Bezeichner für die Verzweigungen zu verwenden.

Um die genaue Korrespondenz zwischen einem bestimmten Tableau und einer bestimmten Klauselmenge herzustellen, wird noch eine Verzweigungsordnung benötigt. Im Tableau soll für zwei Verzweigungsatome p und q gelten, daß, falls $p{>_K}q$ gilt, die mit p markierte Verzweigung im Tableau vor der mit q markierten stehen soll.

Wie in den folgenden Abschnitten eingeführt wird, wird ein Tableau[1] T mit

$$T = \langle\emptyset, ((j, \langle\{a\}, \emptyset\rangle), (\overline{j}, \langle\{b\}, \emptyset\rangle))\rangle$$

in eine Klauselemenge $K = \{\{\overline{j}, a\}, \{j, b\}\}$ transformiert und umgekehrt. Da die Verzweigungsliterale im Tableau nur zur Bezeichnung verwendet werden, wäre es auch denkbar, daß T in die Klauselmenge $K' = \{\{j, a\}, \{\overline{j}, b\}\}$ übersetzt wird. Diese Variante wird jedoch nicht verwendet, da die Klauselmenge der Bedeutung nach aus $(j \to a) \wedge (\neg j \to b)$ hergeleitet werden soll, was zur Verwendung von K führt.

Es wäre auch möglich, die Verzweigungsliterale im Tableau auf den Ästen zu halten, also wie bei den Klauselmengen als Tableauformeln zu verwenden. Das ist jedoch nicht sinnvoll, da die Verzweigungsliterale keinen Beitrag zum Tableaubeweisverfahren liefern und nur den Suchraum vergrößern würden.

3.3 Transformation von Klauselmengen in Tableaus

Bei dieser Transformation wird ein Tableau aus einer Klauselmenge aufgebaut. Dabei werden die Verzweigungen im Tableau durch die Verzweigungsliterale der Klauseln bestimmt. Die Verzweigungsliterale einer Klausel geben jedoch nicht

[1] Hier sei nochmals, um Verwechslungen zu vermeiden, darauf hingewiesen, daß es sich bei den Multiknoten um Formelmengen handelt. So ist $T = \langle\{a, b\}, \emptyset\rangle$ ein lineares Tableau, bestehend aus den beiden Formeln a und b und nicht etwa aus der Klausel $\{a, b\}$.

immer einen kompletten Pfad von der Wurzel zu einem Blatt im Tableau an. Außerdem ist, da es sich bei den Klauseln um Mengen handelt, keine Reihenfolge, in der die Verzweigungen im Tableau vorkommen sollen, aus den Klauseln ablesbar. Es ist also möglich, aus einer Klauselmenge verschiedene Tableaus zu erzeugen. Dieses Problem kann durch die Verwendung von Verzweigungsordnungen umgangen werden.

Die Transformation einer Klauselmenge K in ein Tableau T unter einer Verzweigungsordnung $>_K$ wird durch die Funktion Φ beschrieben:

$$\Phi : \mathcal{K} \times \mathcal{O} \rightarrow 2^{\mathcal{T}^*}$$

$$\Phi(K, >_K) = \{T | T = \texttt{ctt}(K, >_K)\}$$

Dabei erzeugt Φ eine Klasse von Tableaus, die alle möglichen aus eine Klauselmenge unter Verwendung einer bestimmten Ordnung herstellbaren Tableaus enthält. Mit der Funktion `ctt` können alle in dieser Menge enthaltenen Tableaus berechnet werden. Jedoch wird mit `ctt` immer nur ein Tableau erzeugt. `ctt`, beziehungsweise die von `ctt` aufgerufene Funktion `fe`, enthält sogenannte Wahlstellen (choicepoints), an denen entschieden werden muß, welches Tableau aus der Menge berechnet werden soll. Die Funktion `fe` enthält zwei Stellen, an denen eine solche Wahlmöglichkeit besteht: Direkt nach dem Aufruf von `fe` muß ein bezüglich der Ordnung maximales Element berechnet werden. Existieren mehrere, so muß eines davon ausgewählt werden. Die zweite Möglichkeit für eine Auswahl erhält man, wenn Fall (6) von `fe` zutrifft.

Doch zunächst wird die Funktion `ctt` beschrieben. Die beiden Wahlstellen werden bei der Einführung von `fe` näher betrachtet.

Die Funktion `ctt` berechnet das Tableau auf folgende Weise: Nacheinander, in beliebiger Reihenfolge, werden die Klauseln aus K in ein Tableau eingefügt, die erste Klausel in ein leeres Tableau und die folgenden immer in das davor entstandene. Das wird durch die Funktion

$$\texttt{ctt} : \mathcal{K} \times \mathcal{O} \rightarrow \mathcal{T}$$

$$\texttt{ctt}(K, >_K) = \begin{cases} \langle \emptyset, \emptyset \rangle & (1) \\ \qquad falls\ K = \emptyset & \\ \texttt{fe}(C, >_K, \texttt{ctt}(K \backslash \{C\}, >_K)) & (2) \\ \qquad sonst,\ mit\ C \in K\ beliebig & \end{cases}$$

dargestellt.

Falls die Klauselmenge K leer ist, wird das leere Tableau ausgegeben (1). Sonst wird eine Klausel C aus K ausgewählt und in das rekursiv aus der Klauselmenge $K\backslash\{C\}$ erzeugte Tableau eingefügt. Aus K werden alle Klauseln, also sowohl die aus der starren als auch die aus der universellen Teilmenge, in das Tableau überführt.

Die Reihenfolge, in der die Klauseln in das Tableau eingefügt werden, wird durch die Aufrufe von `ctt` festgelegt. Sie ist beliebig und hat, wie später in Lemma 4.1 gezeigt wird, keinen Einfluß auf das resultierende Tableau.

Die eigentliche Transformation wird mit der Funktion `fe` durchgeführt, die ein Tableau um eine Klausel erweitert. Anhand der Verzweigungsliterale in der Klausel werden die Stellen im Tableau gesucht, an denen die Klausel eingefügt werden soll. Dabei ist die Verzweigungsordnung entscheidend, da mit ihr die Reihenfolge der im Tableau auftretenden Verzweigungen festgelegt wird.

Die Funktion `fe` ist wie folgt definiert:

$$\texttt{fe} : \mathcal{C} \times \mathcal{O} \times \mathcal{T} \to \mathcal{T}$$

$$\texttt{fe}(C, >_K, T) = \begin{cases} \langle W \cup \{\texttt{fma}(C)\}, U\rangle & (1) \\ \qquad für\ n = 0 & \\ \langle W, ((k, T_1), (\overline{k}, \texttt{fe}(C', >_K, T_2)))\rangle & (2) \\ \qquad für\ n > 0,\ U \neq \emptyset,\ j = k & \\ \langle W, ((k, \texttt{fe}(C', >_K, T_1)), (\overline{k}, T_2))\rangle & (3) \\ \qquad für\ n > 0,\ U \neq \emptyset,\ j = \overline{k} & \\ \langle W, ((j, \langle\emptyset, U\rangle), (\overline{j}, \texttt{fe}(C', >_K, \langle\emptyset, U\rangle)))\rangle & (4) \\ \qquad für\ n > 0,\ \rho(j) >_K \rho(k)\ oder\ U = \emptyset & \\ \langle W, ((k, \texttt{fe}(C, >_K, T_1)), (\overline{k}, \texttt{fe}(C, >_K, T_2)))\rangle & (5) \\ \qquad für\ n > 0,\ U \neq \emptyset,\ \rho(k) >_K \rho(j) & \\ Ergebnis\ von\ (4)\ oder\ (5) & (6) \\ \qquad für\ n > 0,\ U \neq \emptyset,\ \rho(j) <>_K \rho(k) & \end{cases}$$

Dabei werden, der besseren Lesbarkeit wegen, die folgenden Abkürzungen verwendet: `fe` erhält als Eingabe die Klausel $C = \{j_1, \ldots, j_n, l_1, \ldots, l_m\}$ und das Tableau $T = \langle W, U\rangle$, wobei $U = ((k, T_1), (\overline{k}, T_2))$ oder $U = \emptyset$ ist. Außerdem ist $j \in max_{>_K}(\{j_1, \ldots, j_n\})$ ein bezüglich der Verzweigungsordnung maximales Element, und $C' = C\backslash\{j\}$ ist die Eingabeklausel ohne dieses maximale Element. Falls es mehrere maximale Elemente in $max_{>_K}(\{j_1, \ldots, j_n\})$ gibt, wird eines davon ausgewählt (choicepoint). Die Wahl kann die Struktur des entstehenden Tableaus beinflussen. Die Auswahl eines der maximalen Elemente ist für den weiteren Aufbau des Tableaus bindend. Neue Vergleiche müssen immer in derselben Weise beantwortet werden. Insbesondere muß für das gewählte maximale

Element j die Ordnung erweitert werden:

$$\forall k \in max_{>_K}(\{j_1, \ldots, j_n\}), k \neq j : \rho(j) >_K \rho(k).$$

Es ergeben sich beim Einfügen der Klausel C in das Tableau T sechs verschiedene Fälle:

Fall (1) ist der Basisfall. Die Klausel K enthält keine Verzweigungsliterale, das heißt, die Klausel wird in eine Tableauformel überführt und an der Wurzel von T eingefügt, also in den Multiknoten W.

Es muß für C die richtige Stelle im Tableau gefunden werden. Diese Stelle ist dann gefunden, wenn alle Verzweigungsliterale in der Klausel auf dem Pfad bis zur Wurzel vorkommen. Auf dem Pfad zur Wurzel können auch Verzweigungsliterale stehen, die nicht in der Klausel vorkommen. Die restlichen Fälle in `fe` kommen also durch den Vergleich der Verzweigungsliterale j und k bezüglich der Ordnung zustande. Dabei ist j eines der maximalen Verzweigungsliterale aus C, und die nächste Verzweigung im Tableau ist mit k und $\overline{k}$ gekennzeichnet. Es gibt die Möglichkeiten $j = k$ (2), $j = \overline{k}$ (3), $\rho(j) >_K \rho(k)$ (4), $\rho(k) >_K \rho(j)$ (5) und $\rho(k) <>_K \rho(j)$ (6).

Bei Fall (2) stimmt das aktuelle Verzweigungsliteral k in T mit dem maximalen j in C überein. Es kann in dem mit $\overline{k}$ gekennzeichneten Teilbaum T_2 von T weitergesucht werden, also $C \backslash \{j\}$ rekursiv in T_2 eingefügt werden. Die Erweiterung von T durch `fe`, Fall (2), ist in Bild 3.2 zu sehen.

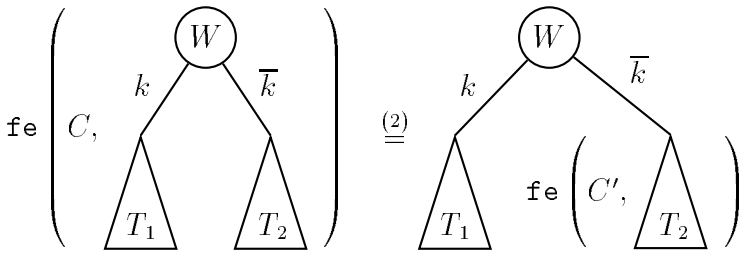


Abbildung 3.2: Erweiterung von T durch Fall (2) von `fe`.

Analog zu (2) verläuft auch Fall (3), jedoch wird in dem mit k gekennzeichneten Teilbaum T_1 von T weitergesucht.

Fall (4) ist der komplizierteste. Es gilt $\rho(j) >_K \rho(k)$, das heißt, die mit j gekennzeichnete Verzweigung kommt in T noch nicht vor. Sonst könnte im Tableau

die mit k gekennzeichnete Verzweigung nicht die oberste sein oder j nicht in C vorkommen. Es muß eine neue Verzweigung erzeugt werden. Dabei muß der Multiknoten W aus T als Multiknoten der neuen Verzweigung verwendet und in T durch den leeren Multiknoten ersetzt werden. T, mit leerem Multiknoten, wird dann sowohl als linker als auch als rechter Teilbaum der neuen Verzweigung benutzt. Die beiden Teilbäume werden mit j und mit $\overline{j}$ gekennzeichnet. Schließlich wird $C\backslash\{j\}$ rekursiv in den mit j gekennzeichneten Teilbaum eingefügt. In Bild 3.3 wird T durch Fall (4) von `fe` erweitert:

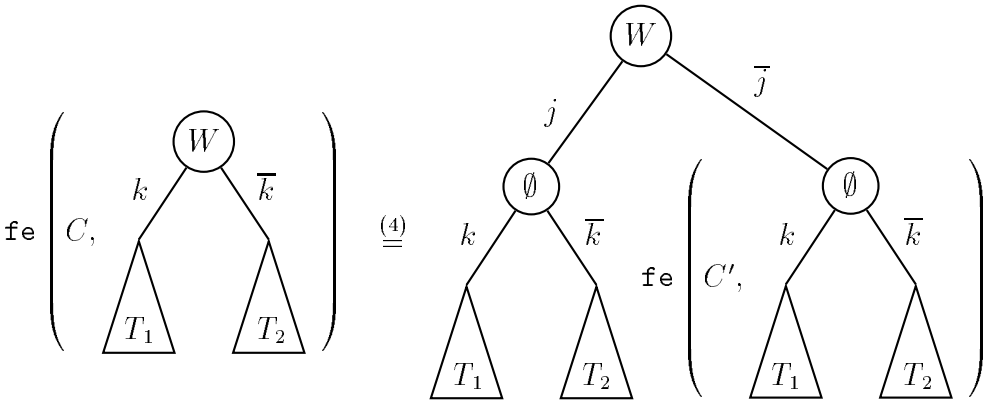


Abbildung 3.3: Erweiterung von T durch Fall (4) von `fe`.

In Fall (5) ist j, eines der maximalen Verzweigungsliterale aus C, kleiner als das aktuelle Verzweigungsliteral k in T. k kommt in C nicht vor. Weil $\rho(k) >_K \rho(j)$ gilt, wäre es spätestens jetzt als maximales Verzweigungsliteral gewählt worden. Also wird C rekursiv in beide Teilbäume von T eingefügt. Fall (5) wird in Bild 3.4 gezeigt.

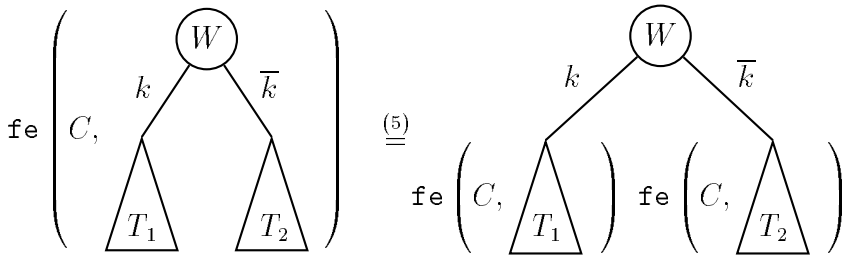


Abbildung 3.4: Erweiterung von T durch Fall (5) von `fe`.

Der letzte Fall (6) tritt nur auf, wenn die Ordnung $>_K$ nicht total ist. Dann ist es möglich, daß $\rho(l) <>_K \rho(k)$ gilt, und man kann wählen (choicepoint), ob

$\rho(j) >_K \rho(k)$ oder $\rho(k) >_K \rho(j)$ gelten soll. Die Wahl muß jedoch endgültig sein, und weitere Vergleiche zwischen j und k müssen immer gleich beantwortet werden. Es kann also wie in (4) oder (5) vorgegangen werden.

Nun fehlt noch die Übersetzung von Klauseln in Tableauformeln. Dazu wird die Funktion

$$\texttt{fma} : \mathcal{C} \to \mathcal{F}$$

$$\texttt{fma}(\{l_1, \ldots, l_n\}) = \begin{cases} \bot & (1) \\ \quad falls\ n = 0 & \\ l_1 \vee \ldots \vee l_n & (2) \\ \quad falls\ \{l_1, \ldots, l_n\}\ aus\ der\ starren\ Teilmenge & \\ \forall\, x_1, \ldots, x_h\ l_1 \vee \ldots \vee l_n & (3) \\ \quad falls\ \{l_1, \ldots, l_n\}\ aus\ der\ universellen\ Teilm. & \end{cases}$$

verwendet. $x_1, \ldots, x_h$ sind die freien Variablen in $\{l_1, \ldots, l_n\}$.

Der schrittweise Aufbau eines Tableaus aus eine Klauselmenge wird im folgenden Beispiel verdeutlicht:

Beispiel 3.1
Aus der Klauselmenge

$$K = \{\{\neg j_1, a\}, \{j_1, b\}, \{\neg j_1, \neg j_3, e\}, \{\neg j_1, \neg j_2, c\}, \{\neg j_1, j_2, d\}, \{\neg j_1, j_3, f\}\}$$

wird das Tableau $T = \texttt{ctt}(K, >_K)$ erzeugt. K enthält die Verzweigungsliterale j_1, $\neg j_1$, j_2, $\neg j_2$, j_3, $\neg j_3$ und die Formelliterale a, b, c, d, e, f. Als Ordnungsrelation auf den Verzweigungsatomen j_1, j_2, j_3 wird die totale Ordnung $>_K$ verwendet. Es gilt $j_1 >_K j_2$, $j_1 >_K j_3$ und $j_2 >_K j_3$.

Durch `ctt` werden die Klauseln aus K nacheinander mit `fe` in das jeweils im vorherigen Schritt entstandene Tableau eingefügt. Anstelle von $\texttt{fe}(C, >_K, T)$ wird der besseren Lesbarkeit wegen im Beispiel $\texttt{fe}(C, T)$ verwendet.

Als erste Klausel wird $\{\neg j_1, a\}$ in das leere Tableau eingefügt.

$$\begin{aligned} &\texttt{fe}(\{\neg j_1, a\}, \langle \emptyset, \emptyset \rangle) \\ &\qquad \overset{(4)}{=} \langle \emptyset, ((\neg j_1, \langle \emptyset, \emptyset \rangle), (j_1, \texttt{fe}(\{a\}, \langle \emptyset, \emptyset \rangle))) \rangle \\ &\qquad \overset{(1)}{=} \langle \emptyset, ((\neg j_1, \langle \emptyset, \emptyset \rangle), (j_1, \langle \{a\}, \emptyset \rangle)) \rangle = T_1 \end{aligned}$$

Es trifft Fall (4) in `fe` zu, da im leeren Tableau $U = \emptyset$ gilt. Anstelle des leeren Unterbaums wird eine neue Verzweigung für j_1 erzeugt und a rekursiv ($n = 0$,

Fall (1)) in den mit j_1 gekennzeichneten Teilbaum eingefügt.

Als nächstes wird die Klausel $\{j_1, b\}$ in das neu entstandene Tableau T_1 eingefügt.

$$\begin{aligned}
&\texttt{fe}(\{j_1, b\}, T_1) \\
&\qquad \overset{(3)}{=} \quad \langle \emptyset, ((\neg j_1, \texttt{fe}(\{b\}, \langle \emptyset, \emptyset \rangle)), (j_1, \langle \{a\}, \emptyset \rangle)) \rangle \\
&\qquad \overset{(1)}{=} \quad \langle \emptyset, ((\neg j_1, \langle \{b\}, \emptyset \rangle), (j_1, \langle \{a\}, \emptyset \rangle)) \rangle = T_2
\end{aligned}$$

Jetzt ist die Verzweigung für j_1 schon vorhanden, so daß mit $\neg j_1$ abgestiegen ($j = \overline{k}$, Fall (3)) und b eingefügt werden kann ($n = 0$, Fall (1)). Die beiden resultierenden Tableaus, T_1 und T_2, sind in Bild 3.5 dargestellt.

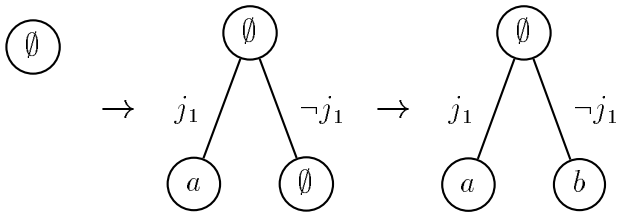


Abbildung 3.5: Das leere Tableau und T_1 und T_2 nach dem Einfügen von $\{\neg j_1, a\}$ und $\{j_1, b\}$.

Beim Einfügen der Klauseln $\{\neg j_1, \neg j_3, e\}$ und $\{\neg j_1, \neg j_2, c\}$ entsteht jeweils eine neue Verzweigung (in Fall (4)).

$$\begin{aligned}
\texttt{fe}(\{\neg j_1, \neg j_3, e\}, T_2) &\quad \overset{(2)(4)(1)}{=} \quad T_3 \\
\texttt{fe}(\{\neg j_1, \neg j_2, c\}, T_3) &\quad \overset{(2)(4)(1)}{=} \quad T_4
\end{aligned}$$

Bei $\{\neg j_1, \neg j_2, c\}$ wird die neue Verzeigung für j_2 nicht an einem Blatt des Baums, sondern oberhalb der mit j_3 gekennzeichneten Verzweigung hinzugefügt. Dabei wird der Teilbaum mit j_3 an jeden der neuen Äste angehängt. T_3 und T_4 sind in Bild 3.6 zu sehen.

Für die Klauseln $\{\neg j_1, j_2, d\}$ und $\{\neg j_1, j_3, f\}$ müssen keine neuen Verzweigungen erzeugt werden.

$$\begin{aligned}
\texttt{fe}(\{\neg j_1, j_2, d\}, T_4) &\quad \overset{(2)(3)(1)}{=} \quad T_5 \\
\texttt{fe}(\{\neg j_1, j_3, f\}, T_5) &\quad \overset{(2)(5)2\times((3)(1))}{=} \quad T
\end{aligned}$$

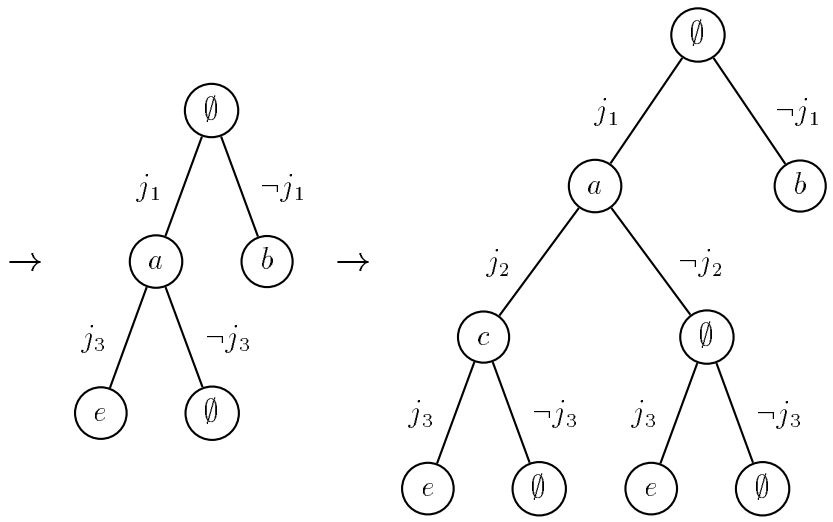


Abbildung 3.6: T_3 und T_4 nach dem Einfügen von $\{\neg j_1, \neg j_3, e\}$ und $\{\neg j_1, \neg j_2, c\}$.

In beiden Fällen wird entsprechend den Verzweigungsliteralen im Tableau abgestiegen, bis die Stellen gefunden sind, an denen d und f in das Tableau eingefügt werden. Bei $\{\neg j_1, j_3, f\}$ muß im zweiten Schritt (Fall (5)) in beide Tableauäste abgestiegen werden, da in diesem Fall $\rho(k) >_K \rho(j)$ ($j_2 >_K j_3$) gilt. T_5 und das Ergebnistableau T sind – in Bild 3.7 – ebenfalls als Baum abgebildet. □

Hier soll noch darauf hingewiesen werden, daß man bei der Transformation von Klauselmengen in Tableaus erst nach dem Einfügen aller Klauseln ein der Definition entsprechendes Tableau erhält. Zum Beispiel stellt die Klausel $\{\neg j, a\}$ nur den mit $\neg j$ markierten Teil einer Verzweigung dar. Beim Einfügen bleibt der mit j markierte Teil leer. Deshalb müssen alle Klauseln eingefügt werden, damit ein sinnvolles Tableau entsteht.

3.4 Transformation von Tableaus in Klauselmengen

Die Transformation eines Tableaus T in eine Klauselmenge C ist einfacher als die in umgekehrter Richtung, da in der Klauselmenge keine Ordnung zu beachten ist. Es werden für jeden Multiknoten W in T die Verzweigungsprädikate $j_1, \ldots, j_n$ auf dem Pfad von der Wurzel nach W bestimmt und für jede Tableauformel $l_1 \vee \ldots \vee l_m$ in W eine Klausel $\{j_1, \ldots, j_n, l_1, \ldots, l_m\}$ gebildet. Dabei wird auch eine Ordnung erzeugt, unter der aus der Ergebnisklauselmenge wieder das ursprüngliche Tableau hergestellt werden kann.

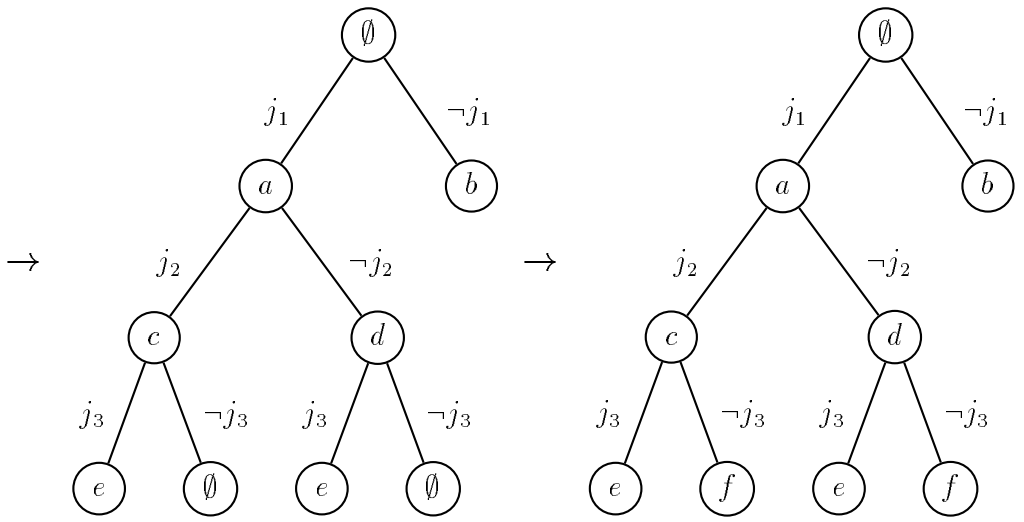


Abbildung 3.7: T_5 und T nach dem Einfügen von $\{\neg j_1, j_2, d\}$ und $\{\neg j_1, j_3, f\}$.

Die Transformation wird durch die Funktion Ψ beschrieben. Dabei ist Ψ definiert als:

$$\Psi : \mathcal{T} \rightarrow \mathcal{K} \times \mathcal{O}$$

Mit $\Psi(T) = \langle K, >_T \rangle$ wird also das Paar[2] bestehend aus einer Klauselmenge K und einer Verzweigungsordnung $>_T$ für K bestimmt. Es wird hier $>_T$ anstelle von $>_K$ verwendet, um deutlich zu machen, daß die Ordnung durch das Tableau T bestimmt wurde.

Die Berechnung der Klauselmenge und der Ordnung werden in der folgenden Beschreibung der besseren Lesbarkeit wegen getrennt voneinander behandelt.

Soll durch Ψ nur die Klauselmenge berechnet werden, so wird vereinfachend $K = \Psi(T)$ verwendet. Für die Berechnung der Klauselmenge K wird Ψ durch

$$\Psi(T) = \texttt{ttc}(\emptyset, T)$$

definiert. Zur Erzeugung der Ordnung wird die Funktion `ord` (s.u.) verwendet, die dann in Fall (2) von `ttc` noch zusätzlich aufgerufen wird.

[2] Die Schreibweise $(\langle , \rangle)$ für das Paar aus Klauselmenge und Ordnung gleicht der Paarschreibweise für Tableaus, die aus einem Multiknoten und Unterbäumen bestehen, ist jedoch nicht mit dieser zu verwechseln.

Doch zunächst wird die Funktion `ttc` erklärt:

$$\texttt{ttc} : \mathcal{C} \times \mathcal{T} \rightarrow \mathcal{K}$$

$$\texttt{ttc}(V,T) = \begin{cases} \texttt{kl}(V,W) & (1) \\ \quad für\ U = \emptyset & \\ \texttt{kl}(V,W) \cup \texttt{lö}(\texttt{ttc}(V \cup \{\overline{k}\}, T_1), \texttt{ttc}(V \cup \{k\}, T_2), k) & (2) \\ \quad sonst & \end{cases}$$

`ttc` erhält als Eingabe ein Tableau $T = \langle W, U \rangle$ und eine Menge von Verzweigungsliteralen V, die bisher auf dem Pfad nach T gesammelt wurden. Für alle Unterbäume U in T gilt $U = ((k, T_1), (\overline{k}, T_2))$ oder $U = \emptyset$.

In Fall (1) von `ttc` werden, da T keine weiteren Unterbäume besitzt, nur die Tableauformeln im Multiknoten W mit der Funktion `kl` in Klauseln gewandelt.

Sind noch Unterbäume vorhanden, Fall (2), dann wird `ttc` rekursiv auf die beiden Unterbäume angewendet. Die Klauselmenge, bestehend aus der Vereinigung der zu W gehörenden Klauseln und der aus den Unterbäumen erzeugten Klauseln, wird als Ergebnis zurückgegeben. Auf die Ergebnisklauselmengen der beiden Unterbäume wird noch die Funktion `lö` angewendet, um doppelte, aus den beiden Teilbäumen erzeugte, Klauseln zu beseitigen.

Neben der Klauselmenge soll aus dem Tableau auch eine Ordnung erzeugt werden. Die Bestimmung dieser Ordnung wurde jedoch bewußt nicht in der Funktion `ttc` formuliert, da die Berechnung der Klauselmenge sonst sehr unübersichtlich wird. Die Ordnung wird ebenfalls beim Durchlauf durch das Tableau bestimmt. Sie wird, beginnend mit der leeren Ordnung, jeweils bei der Ausführung von Schritt (2) in `ttc` erweitert. Dies geschieht mit der Funktion `ord`:

$$\texttt{ord} : \mathcal{C} \times \mathcal{L} \times \mathcal{O} \rightarrow \mathcal{O}$$

$$\texttt{ord}(V, k, >_T) = \forall j \in V : erweitere >_T um\ \rho(j) >_T \rho(k)$$

Die Funktion `kl` ist auf folgende Weise definiert:

$$\texttt{kl} : \mathcal{C} \times \mathcal{W} \rightarrow \mathcal{K}$$

$$\texttt{kl}(V,W) = \begin{cases} \emptyset & (1) \\ \qquad falls\ W = \emptyset & \\ \{V\} \cup \texttt{kl}(V, W\backslash\{F\}) & (2) \\ \qquad falls\ F \in W\ und\ F = \bot & \\ \{V \cup \{l_1, \ldots, l_m\}\} \cup \texttt{kl}(V, W\backslash\{F\}) & (3) \\ \qquad falls\ F \in W\ und\ F = l_1 \vee \ldots \vee l_m & \\ \{V \cup \{l_1, \ldots, l_m\}\} \cup \texttt{kl}(V, W\backslash\{F\}) & (4) \\ \qquad falls\ F \in W\ und\ F = \forall\ x_1, \ldots, x_h\ l_1 \vee \ldots \vee l_n & \end{cases}$$

In Fall (2) wird die leere Klausel $\Box$ zurückgegeben, falls V eine leere Menge ist. Außerdem ist zu beachten, daß die in Fall (3) erzeugten Klauseln in die starre und die in Fall (4) erzeugten Klauseln in die universelle Teilmenge der zu erzeugenden Klauselmenge eingefügt werden. Die Klauseln aus Fall (2) werden immer in die starre Teilmenge genommen.

Nun bleibt noch die Definition von `lö` zu klären.

$$\texttt{lö} : \mathcal{K} \times \mathcal{K} \times \mathcal{L} \rightarrow \mathcal{K}$$

$$\texttt{lö}(K_1, K_2, k) = \begin{cases} \texttt{lö}(K_1\backslash\{C_1\} \cup \{C_1\backslash\{k, \overline{k}\}\}, K_2\backslash\{C_2\}, k) & (1) \\ \qquad falls\ es\ C_1 \in K_1\ und\ C_2 \in K_2\ gibt\ mit & \\ \qquad C_1\backslash\{k,\overline{k}\} = C_2\backslash\{k,\overline{k}\}\ und\ k \notin min_{>_T}(C_1) \cup min_{>_T}(C_2) & \\ K_1 \cup K_2 & (2) \\ \qquad sonst & \end{cases}$$

Mit `lö` werden in den beiden Klauselmengen K_1 und K_2 die Klauseln gelöscht, die aus redundanten Teilbäumen stammen. Durch die Bedingung $k \notin min_{>_T}(C_1) \cup min_{>_T}(C_2)$ in Fall (1) von `lö` wird verhindert, daß das Tableau $\langle\emptyset, ((j, \langle\{a\}, \emptyset\rangle), (\overline{j}, \langle\{a\}, \emptyset\rangle))\rangle$ bei der Ausführung von `ttc` auf die Klauselmenge $\{\{a\}\}$ reduziert wird.

Am besten läßt sich die Wirkung von `lö` beim Vergleich der Klauselmengen zeigen, die mit und ohne die Verwendung von `lö` erzeugt werden. Dazu wird folgendes Beispiel betrachtet:

Beispiel 3.2
In Beispiel 3.1 wurde das Tableau $T = \texttt{ctt}(K, >_K)$ aus der Klauselmenge

$$K = \{\{\neg j_1, a\}, \{j_1, b\}, \{\neg j_1, \neg j_3, e\}, \{\neg j_1, \neg j_2, c\}, \{\neg j_1, j_2, d\}, \{\neg j_1, j_3, f\}\}$$

erzeugt. T (Bild 3.8) soll nun mit Ψ wieder in eine Klauselmenge $K' = \Psi(T) = \texttt{ttc}(\emptyset, T)$ überführt werden.

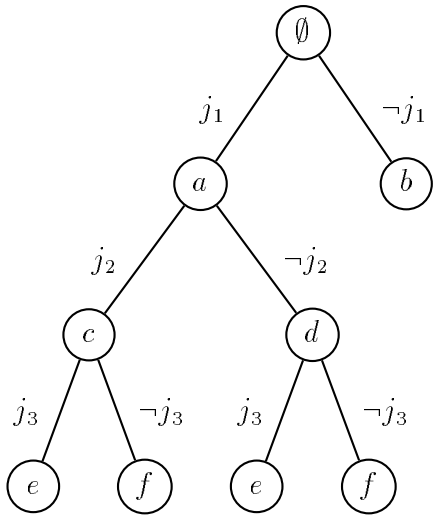


Abbildung 3.8: Das Tableau T.

Zunächst wird K' ohne Ausführung von `lö` berechnet. Dazu wird `ttc` in Fall (2) geändert und `ttc'` verwendet:

$$\texttt{ttc}' : \mathcal{C} \times \mathcal{T} \rightarrow \mathcal{K}$$

$$\texttt{ttc}'(V,T) = \begin{cases} \texttt{kl}(V,W) & \textit{für } U = \emptyset \\ \texttt{kl}(V,W) \cup \texttt{ttc}'(V \cup \{\overline{k}\}, T_1) \cup \texttt{ttc}'(V \cup \{k\}, T_2) & \textit{sonst} \end{cases}$$

$K' = \texttt{ttc}'(\emptyset, T)$ wird rekursiv auf folgende Weise erzeugt:

$$\texttt{ttc}'(\emptyset, T) = \emptyset \cup \texttt{ttc}'(\{\neg j_1\}, T_1) \cup \texttt{ttc}'(\{j_1\}, \langle\{b\}, \emptyset\rangle)$$

Dabei sind:

$$\begin{aligned} \texttt{ttc}'(\{j_1\}, \langle\{b\}, \emptyset\rangle) &= \{\{j_1, b\}\} \\ \texttt{ttc}'(\{\neg j_1\}, T_1) &= \{\{\neg j_1, a\}\} \cup \texttt{ttc}'(\{\neg j_1, \neg j_2\}, T_2) \cup \texttt{ttc}'(\{\neg j_1, j_2\}, T_3) \end{aligned}$$

Die Klauseln für die Teilbäume T_2 und T_3 erhält man analog:

$$\begin{aligned} \texttt{ttc}'(\{\neg j_1, \neg j_2\}, T_2) &= \{\{\neg j_1, \neg j_2, c\}, \{\neg j_1, \neg j_2, \neg j_3, e\}, \{\neg j_1, \neg j_2, j_3, f\}\} \\ \texttt{ttc}'(\{\neg j_1, j_2\}, T_3) &= \{\{\neg j_1, j_2, d\}, \{\neg j_1, j_2, \neg j_3, e\}, \{\neg j_1, j_2, j_3, f\}\} \end{aligned}$$

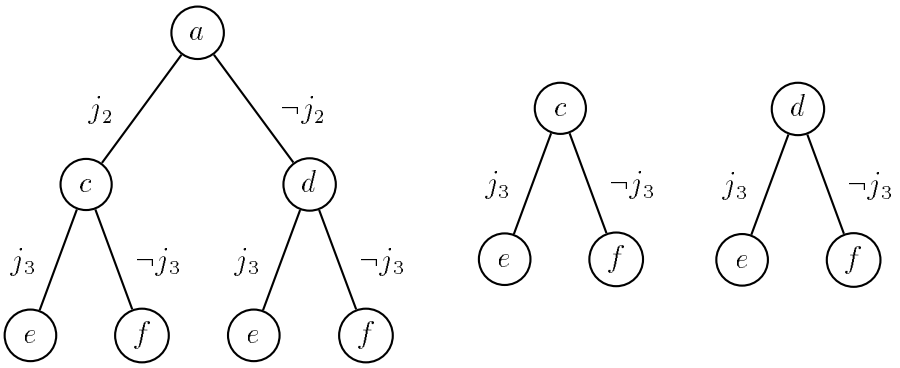


Abbildung 3.9: Die Teilbäume T_1, T_2 und T_3.

Bild 3.9 zeigt die Teilbäume T_1, T_2 und T_3.

Das Ergebnis ist die Klauselmenge

$$\begin{aligned} K' \;=\; & \{\{j_1, b\}, \{\neg j_1, a\}, \{\neg j_1, \neg j_2, c\}, \{\neg j_1, \neg j_2, \neg j_3, e\}, \{\neg j_1, \neg j_2, j_3, f\}, \\ & \{\neg j_1, j_2, d\}, \{\neg j_1, j_2, \neg j_3, e\}, \{\neg j_1, j_2, j_3, f\}\}, \end{aligned}$$

die jedoch nicht mit der ursprünglichen Klauselmenge K, aus der T erzeugt wurde, übereinstimmt. Durch die redundanten Teilbäume in T werden mehr Klauseln erzeugt als gewünscht.

Die Berechnung von $K'' = \texttt{ttc}(\emptyset, T)$ unter Verwendung von `lö` in `ttc` verläuft analog zu der oben für K' beschriebenen. Die Funktion `lö` wird jeweils auf die beiden aus den Teilbäumen erzeugten Klauselmengen angewendet. Im Beispiel tritt Fall (1) von `lö` nur bei den aus den Teilbäumen T_2 und T_3 zurückgegebenen Klauselmengen auf.

$$\begin{aligned} & \texttt{lö}(\{\{\neg j_1, \neg j_2, c\}, \underline{\{\neg j_1, \neg j_2, \neg j_3, e\}}, \{\neg j_1, \neg j_2, j_3, f\}\}, \\ & \qquad \{\{\neg j_1, j_2, d\}, \underline{\{\neg j_1, j_2, \neg j_3, e\}}, \{\neg j_1, j_2, j_3, f\}\}, j_2) \\ \overset{(1)}{=} \; & \texttt{lö}(\{\{\neg j_1, \neg j_2, c\}, \{\neg j_1, \neg j_3, e\}, \underline{\{\neg j_1, \neg j_2, j_3, f\}}\}, \\ & \qquad \{\{\neg j_1, j_2, d\}, \underline{\{\neg j_1, j_2, j_3, f\}}\}, j_2) \\ \overset{(1)}{=} \; & \texttt{lö}(\{\{\neg j_1, \neg j_2, c\}, \{\neg j_1, \neg j_3, e\}, \{\neg j_1, j_3, f\}\}, \{\{\neg j_1, j_2, d\}\}, j_2) \\ \overset{(2)}{=} \; & \{\{\neg j_1, \neg j_3, e\}, \{\neg j_1, \neg j_2, c\}, \{\neg j_1, j_2, d\}, \{\neg j_1, j_3, f\}\} \end{aligned}$$

Beim ersten Aufruf von `lö` gilt $C_1 \backslash \{j_2, \neg j_2\} = C_2 \backslash \{j_2, \neg j_2\}$ für die Klauseln $C_1 = \{\neg j_1, \neg j_2, \neg j_3, e\}$ und $C_2 = \{\neg j_1, j_2, \neg j_3, e\}$. Im zweiten Schritt gilt das gleiche für die Klauseln $\{\neg j_1, \neg j_2, j_3, f\}$ und $\{\neg j_1, j_2, j_3, f\}$. Im letzten Schritt

erfüllen keine zwei Klauseln die Bedingung (1) in `lö`, und die Ergebnisklauselmenge wird zurückgegeben.

Unter Verwendung von `lö` erhält man also die Klauselmenge

$$K'' = \{\{\neg j_1, a\}, \{j_1, b\}, \{\neg j_1, \neg j_3, e\}, \{\neg j_1, \neg j_2, c\}, \{\neg j_1, j_2, d\}, \{\neg j_1, j_3, f\}\},$$

die mit der ursprünglichen Klauselmenge K übereinstimmt.

Die Ordnung ergibt sich durch die Aufrufe der Funktion `ord`. Und zwar durch:

$$\begin{aligned} \texttt{ord}(\{\neg j_1\}, j_2, >_T) &= \{j_1 >_T j_2\} \\ \texttt{ord}(\{\neg j_1, \neg j_2\}, j_3, >_T) &= \{j_1 >_T j_3, j_2 >_T j_3\} \end{aligned}$$

Alle weiteren Aufrufe von `ord` ergeben keine neuen Vergleiche. Damit ist $j_1 >_T j_2$, $j_1 >_T j_3$, $j_2 >_T j_3$ die aus T erzeugte Ordnung. □

4 Eigenschaften der Transformationen

Im letzten Kapitel wurden die Transformationen zwischen Klauselmengen mit Verzweigungsliteralen und Tableaus mit Verzweigungsliteralen eingeführt. Die Transformation von Klauselmengen in Tableaus wird mit der Funktion

$$\Phi : \mathcal{K} \times \mathcal{O} \to [\mathcal{T}]$$

durchgeführt, die von Tableaus in Klauselmengen mit

$$\Psi : \mathcal{T} \to \mathcal{K} \times \mathcal{O}.$$

Nun werden diese Transformationen näher untersucht. Zunächst wird in Abschnitt 4.1 auf die Rolle der Verzweigungsordnung eingegangen. In Abschnitt 4.2 wird gezeigt, daß die Ergebnisse der Transformationen auch wirklich den Defininitionen entsprechende Tableaus und Klauselmengen sind. Der darauf folgende Abschnitt 4.3 ist der Umkehrbarkeit der beiden Transformationen gewidmet. Schließlich werden in Abschnitt 4.4 noch weitere Aussagen angegeben, die sich aus den bis dahin gezeigten Sätzen schließen lassen.

Im Zusammenhang mit der Transformation von Klauseln in Tableauformeln und der Transformation im umgekehrter Richtung wird im folgenden, vor allem in den Beweisen, nicht immer explizit zwischen Klauseln aus der universellen oder der starren Teilmenge einer Klauselmenge beziehungsweise allquantifizierten Formeln oder Formeln mit freien Variablen unterschieden.

Außerdem wird, falls es nicht notwendig ist, nicht immer darauf hingewiesen, daß die Literale Variablen enthalten können. Wenn zum Beispiel von der Transformation der Klausel $C = \{p\}$ in die Tableauformel p gesprochen wird, dann kann das Literal p in C Variablen enthalten. Kommt C in der universellen Teilmenge einer Klauselmenge vor, dann ist die korrespondierende Tableauformel allquantifiziert. Falls C Element der starren Teilmenge ist, so enthält die Tableauformel freie Variablen.

4.1 Eigenschaften der Verzweigungsordnung

Verzweigungsordnungen werden immer nur auf den Verzweigungsliteralen angegeben, die auch in der Klauselmenge vorkommen. Verzweigungsliterale, die nicht

in der Klauselmenge K vorkommen, können in der Ordnung auch verwendet werden, bewirken jedoch bei der Transformation Φ keine Änderung gegenüber Ordnungen, die nur die Verzweigungsliterale aus K enthalten. So kann bei der Transformation einer Klauselmenge, die keine Verzweigungsliterale enthält, zum Beispiel jede beliebige Verzweigungsordnung verwendet werden. Es entsteht aber immer das gleiche Tableau, bei dem alle Formeln an der Wurzel stehen und das keine Unterbäume besitzt. Im folgenden werden für K also immer Ordnungen verwendet, die nur die in K enthaltenen Verzweigungsliterale ordnen. Für die eben beschriebene Klauselmenge, die keine Verzweigungsliterale enthält, kommt dann nur die leere Ordnung in Frage.

4.1.1 Totale Verzweigungsordnungen

Bisher wurde eine Verzweigungsordnung bei der Transformation von Klauselmengen in Tableaus nur unter dem Einfluß, den sie auf die Struktur des entstehenden Tableaus ausübt, betrachtet. Da es möglich ist, mit einer Ordnung verschiedene Tableaus aus einer Klauselmenge aufzubauen, soll nun untersucht werden, unter welchen Bedingungen mehrere oder nur ein Tableau erzeugt werden können.

Falls die Verzweigungsordnung $>_K$ total ist, läßt sich nur ein einziges Tableau aus der Klauselmenge K bilden. In Satz 4.2 wird gezeigt, daß diese Behauptung stimmt. Doch zunächst wird noch das folgende, für den Beweis benötigte Lemma betrachtet:

Lemma 4.1
Bei der Transformation einer Klauselmenge K in ein Tableau hat die Reihenfolge, in der die Klauseln durch die Funktion `ctt` *in das entstehende Tableau eingefügt werden, keinen Einfluß auf das Ergebnistableau.*

Beweis:
$>_K$ sei eine Verzweigungsordnung für K. Die einzelnen Klauseln werden mit der Funktion `fe` zu einem Tableau zusammengebaut. Es bleibt also zu zeigen, daß für zwei Klauseln $C_1 \in K$ und $C_2 \in K$ und ein Tableau T gilt:

$$\texttt{fe}(C_2, >_K, \texttt{fe}(C_1, >_K, T)) = \texttt{fe}(C_1, >_K, \texttt{fe}(C_2, >_K, T))$$

Der Beweis erfolgt mittels Induktion über n_1 und n_2, die Anzahl der Verzweigungsliterale in C_1 und C_2. Dabei sind $j_1 \in max_{>_K}(C_1)$ und $j_2 \in max_{>_K}(C_2)$ maximale Elemente von C_1 und C_2. Die Wahl von j_1 und j_2 kann Einfluß auf die Struktur des entstehenden Tableaus haben. Sind j_1 und j_2 jedoch gewählt, dann hat die Reihenfolge, in der C_1 und C_2 in das Tableau eingefügt werden, keinen Einfluß auf die Struktur des Ergebnistableaus.

Falls $max_{>_K}(C_1)$ mehr als ein Element enthält, beeinflußt die Wahl von j_1 auch die Ordnung $>_K$. Während des weiteren Aufbaus von T gilt, daß j_1 bezüglich $>_K$ größer ist als alle weiteren Verzweigungsliterale aus $max_{>_K}(C_1)$ (siehe Abschnitt 3.3). Diese neuen Vergleiche sind bei der Wahl von j_2 zu berücksichtigen. Falls j_2 zuerst gewählt wird, gilt alles analog.

Außerdem sind $C_1' = C_1 \backslash \{j_1\}$ und $C_2' = C_2 \backslash \{j_2\}$ die Klauseln ohne die gewählten maximalen Elemente, und $T = \langle W, U \rangle$ ist ein Tableau mit $U = ((h, T_1), (\overline{h}, T_2))$.

Bei den folgenden Berechnungen geben die Zahlen über den Gleichheitszeichen den Fall von `fe` an, der bei der entsprechenden Gleichung verwendet wird, und bei `fe` wird die Angabe von $>_K$ weggelassen.

Induktionsanfang: $n_1 = 0$ und $n_2 = 0$

Es kommen keine Verzweigungsliterale in C_1 und C_2 vor. Das heißt, die mit den Klauseln korrespondierenden Formeln werden an der Wurzel von T eingefügt:

$$\begin{aligned} &\texttt{fe}(C_2, \texttt{fe}(C_1, T)) \\ &\quad \overset{(1)(1)}{=} \langle W \cup \{\texttt{fma}(C_1)\} \cup \{\texttt{fma}(C_2)\}, U \rangle \\ &\quad \overset{(1)(1)}{=} \texttt{fe}(C_1, \texttt{fe}(C_2, T)) \end{aligned}$$

Induktionsschritt: $n_2 \to n_2 + 1$

Der Induktionsschritt muß für jede Kombination von j_1, j_2 und h bezüglich $>_K$ durchgeführt werden. Exemplarisch wird hier der Fall $j_1 = h$, $\rho(j_2) >_K \rho(j_1)$ gezeigt.

Zunächst wird die linke Seite der Gleichung expandiert:

$$\begin{aligned} &\texttt{fe}(C_2, \underline{\texttt{fe}}(C_1, T)) \\ &\quad \overset{(2)}{=} \underline{\texttt{fe}}(C_2, \langle W, ((h, T_1), (\overline{h}, \texttt{fe}(C_1', T_2))) \rangle) \\ &\quad \overset{(4)}{=} \langle W, ((j_2, \langle \emptyset, ((h, T_1), (\overline{h}, \texttt{fe}(C_1', T_2))) \rangle), \\ &\qquad\qquad (\overline{j_2}, \texttt{fe}(C_2', \langle \emptyset, ((h, T_1), (\overline{h}, \texttt{fe}(C_1', T_2))) \rangle)) \rangle \rangle \\ &\quad = S \end{aligned}$$

Für die rechte Seite der zu beweisenden Behauptung ergibt sich folgendes:

$$
\begin{aligned}
&\mathtt{fe}(C_1, \underline{\mathtt{fe}}(C_2, T)) \\
&\quad \overset{(4)}{=} \underline{\mathtt{fe}}(C_1, \langle W, ((j_2, \langle \emptyset, ((h, T_1), (\overline{h}, T_2))\rangle), \\
&\qquad\qquad (\overline{j_2}, \mathtt{fe}(C_2', \langle \emptyset, ((h, T_1), (\overline{h}, T_2))\rangle))\rangle)\rangle) \\
&\quad \overset{(5)}{=} \langle W, ((j_2, \underline{\mathtt{fe}}(C_1, \langle \emptyset, ((h, T_1), (\overline{h}, T_2))\rangle)), \\
&\qquad (\overline{j_2}, \mathtt{fe}(C_1, \mathtt{fe}(C_2', \langle \emptyset, ((h, T_1), (\overline{h}, T_2))\rangle))\rangle)\rangle\rangle \\
&\quad \overset{(2)}{=} \langle W, ((j_2, \langle \emptyset, ((h, T_1), (\overline{h}, \mathtt{fe}(C_1', T_2)))\rangle), \\
&\qquad \underline{(\overline{j_2}, \mathtt{fe}(C_1, \mathtt{fe}(C_2', \langle \emptyset, ((h, T_1), (\overline{h}, T_2))\rangle))\rangle)}\rangle\rangle \\
&\quad \overset{IH}{=} \langle W, ((j_2, \langle \emptyset, ((h, T_1), (\overline{h}, \mathtt{fe}(C_1', T_2)))\rangle), \\
&\qquad (\overline{j_2}, \mathtt{fe}(C_2', \underline{\mathtt{fe}}(C_1, \langle \emptyset, ((h, T_1), (\overline{h}, T_2))\rangle))\rangle)\rangle\rangle \\
&\quad \overset{(2)}{=} \langle W, ((j_2, \langle \emptyset, ((h, T_1), (\overline{h}, \mathtt{fe}(C_1', T_2)))\rangle), \\
&\qquad (\overline{j_2}, \mathtt{fe}(C_2', \langle \emptyset, ((h, T_1), (\overline{h}, \mathtt{fe}(C_1', T_2)))\rangle))\rangle)\rangle \\
&\quad = S
\end{aligned}
$$

Unter Anwendung der Induktionshypothese (IH) ergibt sich für $\mathtt{fe}(C_1, \mathtt{fe}(C_2, T))$ das gleiche Tableau, wie für $\mathtt{fe}(C_2, \mathtt{fe}(C_1, T))$.

Die Berechnungen für die weiteren Kombinationen von j_1, j_2 und h bezüglich $>_K$ und der Induktionsschritt $n_1 \rightarrow n_1 + 1$ verlaufen analog. ■

Die Reihenfolge, in der die Klauseln in ein Tableau zusammengefügt werden, hat also keinen Einfluß auf die Struktur des Ergebnistableaus. Damit läßt sich zeigen, daß durch die Transformation Φ unter Verwendung einer totalen Ordnung $>_K$ genau ein Tableau herstellbar ist.

Satz 4.2
Sei K eine Klauselmenge mit Verzweigungsliteralen und $>_K$ eine totale Verzweigunsordnung für K. Dann enthält die Klasse $[T] = \Phi(K, >_K)$ von Tableaus nur ein Tableau.

Beweis:
Bei der Transformation können nur dann mehrere verschiedene Tableaus entstehen, wenn in der Funktion `fe` der Fall (6) auftritt oder $max_{>_K}$ nicht eindeutig ist. Da $>_K$ eine totale Ordnung ist, ergibt sich bei der Berechnung von $max_{>_K}$ jedoch immer nur ein eindeutiges Verzweigungsliteral. Fall (6) von `fe` wird gewählt, wenn $\rho(j_1) <>_K \rho(j_2)$ gilt, wobei j_1 und j_2 Verzweigungsliterale sind. Falls $>_K$ total

ist, gilt jedoch immer $\rho(j_1) >_K \rho(j_2)$ oder $\rho(j_2) >_K \rho(j_1)$. Fall (6) kann also nicht auftreten. Außerdem hat die Reihenfolge, in der die Klauseln in ein entstehendes Tableau eingefügt werden, keinen Einfluß auf das Ergebnistableau (Lemma 4.1). Damit ist die Transormation eindeutig und die Klasse $[T]$ enthält nur ein eindeutiges Tableau. ■

4.1.2 Klauseltotale und tableaueindeutige Verzweigungsordnungen

Es gibt jedoch auch nicht-totale Verzweigungsordnungen, mit denen sich nur ein Tableau erzeugen läßt. Um diese Ordnungen genauer beschreiben zu können, werden *klauseltotale* und *tableaueindeutige* Verzweigungsordnungen definiert.

Definition 4.3 Klauseltotale Ordnung
Die Verzweigungsordnung $>_K$ heißt klauseltotal für die Klauselmenge K, wenn $>_K$ für jede Klausel $C \in K$ auf den Verzweigungsliteralen $Vl(C)$ aus C total ist.

Definition 4.4 Tableaueindeutige Ordnung
Die Verzweigungsordnung $>_K$ heißt tableaueindeutig für die Klauselmenge K, falls $>_K$ klauseltotal ist und für jede Kombination zweier Klauseln $C_1 \in K$ und $C_2 \in K$ gilt:

- *Entweder ordnet $>_K$ die Menge $Vl(C_1) \cup Vl(C_2)$ total oder*
- *C_1 und C_2 enthalten genau ein komplementäres Verzweigungsliteral, und es gilt für $C_1 = \{j, j_1, \ldots, j_i, j_{i+1}, \ldots, j_n\}$ und $C_2 = \{\overline{j}, k_1, \ldots, k_i, k_{i+1}, \ldots, k_m\}$, daß $j_s = k_s$ für $1 \leq s \leq i$, $\rho(j) >_K \rho(j_r)$ für $i < r \leq n$ und $\rho(\overline{j}) >_K \rho(k_r)$ für $i < r \leq m$.*

Jede totale Ordnung für eine Klauselmenge K ist auch klauseltotal und tableaueindeutig für K. Wie bei totalen Ordnungen läßt sich mit tableaueindeutigen Ordnungen durch die Transformation Φ auch nur genau ein Tableau erzeugen. Es gilt der folgende Satz:

Satz 4.5
Sei K eine Klauselmenge mit Verzweigungsliteralen und $>_K$ eine tableaueindeutige Verzweigungsordnung für K. Dann enthält die Menge $T = \Phi(K, >_K)$ von Tableaus nur ein Tableau.

Beweis:
Der Beweis verläuft analog zum Beweis von Satz 4.2. Die Berechnung des maximalen Verzweigungsliterals der neu einzufügenden Klausel C ist eindeutig, da die Verzweigungsliterale aus C durch $>_K$ total geordnet sind ($>_K$ ist klauseltotal). Fall (6) von `fe` tritt auch bei der Verwendung tableaueindeutiger Ordnungen

nicht auf, da die nicht geordneten Verzweigungsliterale, die diesen Fall hervorrufen könnten, in verschiedenen Teilbäumen des Tableaus vorkommen. ■

Die Erzeugung eines Tableaus unter Verwendung einer tableaueindeutigen Ordnung wird im folgenden Beispiel 4.6 gezeigt.

Beispiel 4.6
Für die Klauselmenge

$$K = \{\{\neg j_1, \neg j_2, a\}, \{\neg j_1, j_2, b\}, \{j_1, \neg j_3, c\}, \{j_1, j_3, d\}\}$$

und die Verzweigungsordnung $>_K$ soll das Tableau $T = \Phi(K, >_K)$ berechnet werden. Für die in K vorkommenden Verzweigungsatome j_1, j_2 und j_3 ist $>_K$ mit $j_1 >_K j_2$ und $j_1 >_K j_3$ gegeben. Die Ordnung $>_K$ ist nicht total, da $j_2 <>_K j_3$ gilt, aber tableaueindeutig. Es entsteht das Tableau T mit

$$\begin{aligned} T \;=\; & \langle \emptyset, ((\overline{j_1}, \langle \emptyset, ((\overline{j_2}, \langle \{a\}, \emptyset \rangle), (j_2, \langle \{b\}, \emptyset \rangle))\rangle), \\ & (j_1, \langle \emptyset, ((\overline{j_3}, \langle \{c\}, \emptyset \rangle), (j_3, \langle \{d\}, \emptyset \rangle))\rangle)\rangle \end{aligned}$$

T ist auch in Abbildung 4.1 zu sehen. □

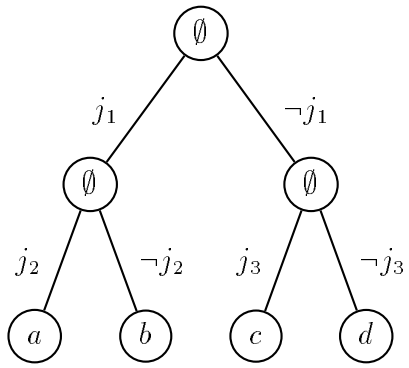


Abbildung 4.1: Das Tableau T.

4.1.3 Beliebige Verzweigungsordnungen

Aus einer beliebigen Verzweigungsordnung $>_K$ können durch Festlegen der Vergleiche für die ungeordneten Verzweigungsliterale verschiedene totale Ordnungen hergestellt werden. Die Menge dieser totalen Ordnungen für eine beliebige Verzweigungsordnung $>_K$ wird mit $\tau(>_K)$ bezeichnet.

Definition 4.7 Totale Ordnungen zu $>_K$
Sei $>_K$ eine beliebige Verzweigungsordnung für die Klauselmenge mit Verzweigungsliteralen K. Mit $\tau(>_K)$ wird die Menge aller totalen Ordnungen bezeichnet, die Spezialisierungen von $>_K$ sind.

Im folgenden Beispiel wird τ für eine Verzweigungsordnung angegeben:

Beispiel 4.8
Für die Klauselmenge K mit den Verzweigungsatomen j_1, j_2 und j_3 ist die Verzweigungsordnung $>_K$ mit den Vergleichen $j_1 >_K j_2$ und $j_1 >_K j_3$ gegeben. Die Menge $\tau(>_K)$ enthält die zwei totalen Ordnungen $>_K^1$ und $>_K^2$ mit

$$\begin{aligned} >_K^1 &= \{j_1 >_K j_2,\ j_1 >_K j_3,\ j_2 >_K j_3\} \\ >_K^2 &= \{j_1 >_K j_2,\ j_1 >_K j_3,\ j_3 >_K j_2\}. \end{aligned}$$

□

In der Menge $\Phi(K, >_K)$ von Tableaus sind alle Tableaus enthalten, die mit Φ aus der Klauselmenge K unter Verwendung einer Ordnung aus $\tau(>_K)$ gebildet werden. Und zwar genau diese und keine weiteren.

Bei der Berechnung eines Tableaus aus der Menge werden die durch $>_K$ nicht geordneten Verzweigungsatome sukzessive geordnet (an den Wahlstellen), so daß mit dem aufgebauten Tableau auch eine tableaueindeutige Verzweigungsordnung entstanden ist. Diese Ordnung muß nicht total sein, da verschiedene Verzweigungsliterale durch die Struktur der Literale in den Klauseln im Tableau in verschiedenen Teilbäumen vorkommen können. Beim Einfügen der Klauseln werden dann keine Vergleiche zwischen diesen Verzweigungsliteralen benötigt. In Beispiel 4.6 wird bei der Berechnung von T in keinem Schritt ein Vergleich zwischen j_2 und j_3 vorgenommen.

Jede totale Verzweigungsordnung für K, die aus einer tableaueindeutigen durch Ordnen der nicht vergleichbaren Verzweigungsliterale entstanden ist, bewirkt, daß mit Φ aus K jeweils das gleiche Tableau erzeugt wird, da mit einer tableaueindeutigen Ordnung immer nur ein eindeutiges Tableau gebildet werden kann (Satz 4.5). Es spielt also keine Rolle, wie die nicht geordneten Verzweigungsliterale geordnet werden. Wird in Beispiel 4.6 die Ordnung um $j_2 >_K j_3$ oder um $j_3 >_K j_2$ erweitert, so erhält man zwei verschiedene totale Ordnungen. Die Anwendung von Φ auf K unter Verwendung dieser totalen Ordnungen ergibt jeweils wieder das Tableau aus Bild 4.1.

Um aus einer Klauselmenge ein eindeutiges Tableau zu bilden, reicht es nicht aus, eine klauseltotale Ordnung zu verwenden. Hier ist es möglich, daß Fall (6) in der

Funktion `fe` auftritt. Zur Verdeutlichung werden in Beispiel 4.9 zwei verschiedene Tableaus unter Verwendung einer klauseltotalen Ordnung gebildet.

Beispiel 4.9
Aus der Klauselmenge

$$K = \{\{\neg j_1, a\}, \{j_1, \neg j_2, b\}, \{j_1, j_2, c\}, \{\neg j_3, d\}, \{j_3, e\}\}$$

und der Verzweigungsordnung $>_K$ sollen die Tableaus aus der Menge $\Phi(K, >_K)$ erzeugt werden. $>_K$ ist klauseltotal und ordnet die Verzweigungsatome aus K auf folgende Weise: $j_1 >_K j_2$ und $j_1 >_K j_3$. Die Ordnung ist nicht tableaueindeutig, da zum Beispiel die Klauseln $\{j_1, j_2, c\}$ und $\{j_3, e\}$ keine der Bedingungen in Definition 4.4 erfüllen. Bei der Berechnung muß der Vergleich zwischen j_2 und j_3 festgelegt werden. Es gibt also zwei mögliche Ergebnistableaus: T_1 mit $j_2 >_K j_3$ und T_2, bei dem $j_3 >_K j_2$ gilt. Beide Tableaus sind in Bild 4.2 zu sehen. □

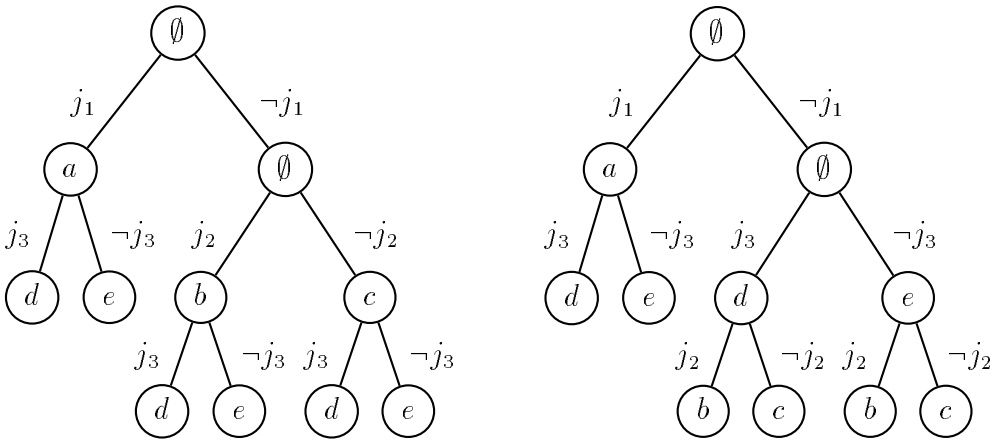


Abbildung 4.2: Die Tableaus T_1 und T_2.

4.1.4 Durch Ψ bestimmte Verzweigungsordnungen

Eine bei der Transformation Ψ von Tableaus in Klauselmengen berechnete Verzweigungsordnung $>_T$ ist tableaueindeutig. Dies ist leicht einzusehen, da durch die Funktion `ord` die Verzweigungsliterale der einzelnen Äste im Tableau jeweils total geordnet werden.

Ähnlich wie bei den eben beschriebenen Ordnungen, die bei der Festlegung der nicht geordneten Verzweigungsatome (an den Wahlstellen) bei der Transformation Φ entstehen, kann $>_T$ auch nur tableaueindeutig sein.

Bei der Rücktransformation des in Bild 4.1 dargestellten Tableaus werden j_2 und j_3 nicht geordnet. Es entsteht die tableaueindeutige Ordnung mit $j_1 >_K j_2$ und $j_1 >_K j_3$.

4.2 Korrektheit der Transformationen

In diesem Abschnitt wird gezeigt, daß die Transformationen auch die gewünschten Tableaus und Klauselmengen als Ergebnis haben.

4.2.1 Korrektheit von Ψ

Zunächst wird für die Transformation von Tableaus in Klauselmengen gezeigt, daß nur Klauselmengen erzeugt werden können, die der Definition für Klauselmengen mit Verzweigungsliteralen entsprechen.

Satz 4.10
Seien F eine Menge von Tableauformeln und T ein Tableau mit Verzweigungsliteralen für F. Dann ist $K = \Psi(T)$ eine Klauselmenge mit Verzweigungsliteralen für F.

Beweis:
Es ist zu zeigen, daß für jedes Tableau T für F gilt, daß $K = \Psi(T)$ eine Klauselmenge für F ist. Der Beweis wird mit Induktion über den Aufbau von T geführt. Zunächst ist jedoch der Sonderfall $F = \emptyset$ zu betrachten.

Falls $F = \emptyset$ gilt, dann gibt es nur ein mögliches Tableau mit Verzweigungsliteralen für F, nämlich $T = \langle \emptyset, \emptyset \rangle$. Zu diesem Tableau wird K auf folgende Weise berechnet:

$$\begin{aligned} K &= \Psi(T) \\ &= \texttt{ttc}(\emptyset, \langle \emptyset, \emptyset \rangle) \\ &= \texttt{kl}(\emptyset, \emptyset) \\ &= \emptyset \end{aligned}$$

Nach Definition 2.17 ist $K = \emptyset$ eine Klauselmenge mit Verzweigungsliteralen für $F = \emptyset$. K kann zum Beispiel mit der trivialen Transformation aus Abschnitt 2.5.3 erzeugt werden.

Im allgemeinen Fall sei $F = \{f_1, \ldots, f_n\}$. Nun wird der Induktionsbeweis über den Aufbau von T geführt:

Induktionsanfang:

Im Basisfall ist $T = \langle\{f_1, \ldots, f_n\}, \emptyset\rangle$ das nur aus einem Multiknoten und keinen Unterbäumen bestehende Tableau (Fall 1 von Definition 2.9). Der Multiknoten enthält dann alle Formeln aus F. Es ergibt sich für $\Psi(T)$:

$$\begin{aligned}\Psi(T) &= \texttt{ttc}(\emptyset, T)\\ &= \texttt{kl}(\emptyset, \{f_1, \ldots, f_n\})\\ &= \{C_1, \ldots, C_n\}\end{aligned}$$

Dabei ist C_i eine Klausel mit Verzweigungsliteralen für f_i für $1 \leq i \leq n$. Die Klauselmenge $\{C_1, \ldots, C_n\}$ kann ebenfalls mit der trivialen Transformation aus F erzeugt werden. $\Psi(T)$ ist also eine Klauselmenge mit Verzweigungsliteralen für F.

Induktionsschritt: $T \rightarrow T'$

$T = \langle W, U\rangle$ ist ein Tableau mit Verzweigungsliteralen für $F = \{f_1, \ldots, f_n\}$ und $K = \Psi(T)$ eine Klauselmenge mit Verzweigungsliteralen für F (Induktinshypothese). T' entsteht durch Erweiterung von T. Es ist zu zeigen, daß $K' = \Psi(T')$ ebenfalls eine Klauselmenge mit Verzweigungsliteralen für F ist. Hier sind drei Fälle zu betrachten, die durch die Definition für Tableaus mit Verzweigungsliteralen (Definition 2.9 in Abschnitt 2.4.2) bestimmt werden.

Die Induktionshypothese (IH) ist $\Psi(T) = K$, wobei T ein Tableau und K eine Klauselmenge mit Verzweigungsliteralen für F sind.

Fall 1: T' entsteht durch eine β-Regel-Erweiterung.

T' entsteht aus T durch Ersetzen der Blätter $B_i = \langle W_i, \emptyset\rangle$ durch

$$B_i' = \langle W_i, ((j, \langle\{\beta_1\}, \emptyset\rangle), (\overline{j}, \langle\{\beta_2\}, \emptyset\rangle))\rangle$$

(Fall 3 von Definition 2.9). Dabei werden mit B_i alle Blätter des Teiltableaus, in dem die β-Formel an der Wurzel steht, bezeichnet. Die Anzahl der Blätter in diesem Teiltableau sei h. Im folgenden gelten, falls nicht anders angegeben, alle Aussagen für $1 \leq i \leq h$.

Bei der Berechnung von $K' = \Psi(T')$ muß, wenn die Abarbeitung des Baums bei einem Blatt B_i' angekommen ist, $\texttt{ttc}(V_i, B_i')$ berechnet werden. V_i ist eine Menge von Verzweigungsliteralen. Dabei enthält die Menge[1] $\overline{V_i}$ genau die Verzweigungs-

[1] Wegen der Korrespondenz von $\langle\emptyset, ((j, \langle\{a\}, \emptyset\rangle), (\neg j, \langle\{b\}, \emptyset\rangle))\rangle$ mit $\{\{\neg j, a\}, \{j, b\}\}$ enthält V_i aus $\texttt{ttc}(V_i, B_i)$ die zu den Verzweigungsliteralen auf dem Pfad nach W_i komplementären Literale. (Es ist $\overline{V} = \{\overline{j_1} \ldots \overline{j_n}\}$ für $V = \{j_1, \ldots, j_n\}$.)

literale auf dem Pfad von der Wurzel nach B_i'. Man erhält:

$$
\begin{aligned}
&\texttt{ttc}(V_i, B_i') \\
&\quad = \quad \underline{\texttt{ttc}}(V_i, \langle W_i, ((j, \langle\{\beta_1\}, \emptyset\rangle), (\overline{j}, \langle\{\beta_2\}, \emptyset\rangle))\rangle) \\
&\quad \overset{(2)}{=} \quad \texttt{kl}(V_i, W_i) \cup \texttt{lö}(\underline{\texttt{ttc}}(V_i \cup \{\overline{j}\}, \langle\{\beta_1\}, \emptyset\rangle), \texttt{ttc}(V_i \cup \{j\}, \langle\{\beta_2\}, \emptyset\rangle), j) \\
&\quad \overset{(1)}{=} \quad \texttt{kl}(V_i, W_i) \cup \texttt{lö}(\underline{\texttt{kl}}(V_i \cup \{\overline{j}\}, \{\beta_1\}), \texttt{ttc}(V_i \cup \{j\}, \langle\{\beta_2\}, \emptyset\rangle), j) \\
&\quad = \quad \texttt{kl}(V_i, W_i) \cup \texttt{lö}(\{V_i \cup \{\overline{j}\} \cup \{\beta_1\}\}, \underline{\texttt{ttc}}(V_i \cup \{j\}, \langle\{\beta_2\}, \emptyset\rangle), j) \\
&\quad \overset{(1)}{=} \quad \texttt{kl}(V_i, W_i) \cup \underline{\texttt{lö}}(\{V_i \cup \{\overline{j}\} \cup \{\beta_1\}\}, \{V_i \cup \{j\} \cup \{\beta_2\}\}, j) \\
&\quad \overset{(2)*}{=} \quad \underline{\texttt{kl}(V_i, W_i)} \cup \{V_i \cup \{\overline{j}\} \cup \{\beta_1\}, V_i \cup \{j\} \cup \{\beta_2\}\} \\
&\quad = \quad \texttt{ttc}(V_i, B_i) \cup \{V_i \cup \{\overline{j}\} \cup \{\beta_1\}, V_i \cup \{j\} \cup \{\beta_2\}\}
\end{aligned}
$$

Vereinfachend werden nun $D_i = \{V_i \cup \{\overline{j}\} \cup \{\beta_1\}\}$ und $E_i = \{V_i \cup \{j\} \cup \{\beta_2\}\}$ verwendet. Beim mit * gekennzeichneten Schritt wird Fall (2) von lö ausgeführt, da $j \in min_>(D_i) \cup min_>(E_i)$ gilt.

Nun ergibt sich für $\Psi(T')$ folgende Berechnung: D_i, E_i und V_i werden weiterhin verwendet.

$$
\begin{aligned}
\Psi(T') &= \texttt{ttc}(\emptyset, T') \\
&= \ldots \cup \texttt{ttc}(V_i, B_i') \cup \ldots \\
&= \ldots \cup (\texttt{ttc}(V_i, B_i) \cup \{D_i, E_i\}) \cup \ldots \\
&\overset{*}{=} (\ldots \cup \texttt{ttc}(V_i, B_i) \cup \ldots) \cup \{D, E\} \\
&= \Psi(T) \cup \{D, E\} \\
&\overset{IH}{=} K \cup \{D, E\} \\
&= K'
\end{aligned}
$$

Im mit * gekennzeichneten Schritt wird die Funktion lö auf die Ergebnisse aus den verschiedenen Teilbäumen angewendet. M sei der Multiknoten, in dem die β-Formel, an der die Verzweigung vorgenommen wurde, vorkommt. Die Klauseln D_i und E_i enthalten jeweils die komplementären Verzweigungsliterale des Pfads von der Wurzel über M nach B_i. In den Klauseln $D_1, \ldots, D_n$ unterscheiden sich also nur die komplementären Verzweigungsliterale der Pfade von M nach B_i (E_i analog). Diese Verzweigungsliterale werden durch die Funktion lö gelöscht, so daß nach der Abarbeitung aller Aufrufe von lö nur noch die Klauseln $D = \{V \cup \{\overline{j}\} \cup \{\beta_1\}\}$ und $E = \{V \cup \{j\} \cup \{\beta_2\}\}$ übrig bleiben. $\overline{V}$ enthält dabei die Verzweigungsliterale des Pfads von der Wurzel nach M.

Bei der weiteren Bearbeitung von lö haben D und E keinen Einfluß auf das Ergebnis. Die Verzweigungsliterale j und $\overline{j}$ kommen ausschließlich in D und E vor, so daß im Zusammenhang mit einer weiteren Klausel und D oder E nie Fall (1) von lö ausgeführt wird.

K' ist eine erweiterte Klauselmenge von K (Fall 3 in Definition 2.16) und damit auch eine Klauselmenge mit Verzweigungsliteralen für F.

Fall 2: T' entsteht durch eine γ-Regel-Erweiterung.

In diesem Fall entsteht T' aus T durch Ersetzen eines Knotens $S = \langle W_1, U_1 \rangle$ durch $S' = \langle W_1 \cup \{\gamma_1\}, U_1 \rangle$ (Fall 2 von Definition 2.9). Bei der Bestimmung von $\Psi(T')$ muß auch $\mathtt{ttc}(V, S')$ berechnet werden. $\overline{V}$ ist dabei wieder die Menge von Verzweigungsliteralen auf dem Pfad von der Wurzel nach S'. Die Berechnung ergibt:

$$\begin{aligned}
\mathtt{ttc}(V, S') &= \mathtt{ttc}(V, \langle W_1 \cup \{\gamma_1\}, U_1 \rangle) \\
&= \mathtt{kl}(V, W_1 \cup \{\gamma_1\}) \cup \mathtt{lö}(\ldots) \\
&= \{V \cup \{\gamma_1\}\} \cup \mathtt{kl}(V, W_1) \cup \mathtt{lö}(\ldots) \\
&= \{V \cup \{\gamma_1\}\} \cup \mathtt{ttc}(V, \langle W_1, U_1 \rangle) \\
&= \{V \cup \{\gamma_1\}\} \cup \mathtt{ttc}(V, S)
\end{aligned}$$

Falls $U_1 = \emptyset$, dann fällt in Zeile 2 und 3 die Berechnung von $\mathtt{lö}(\ldots)$ weg. Vereinfachend wird jetzt $C = \{V \cup \{\gamma_1\}\}$ verwendet.

Es ergibt sich für $\Psi(T')$:

$$\begin{aligned}
\Psi(T') &= \mathtt{ttc}(\emptyset, T') \\
&= \ldots \cup \mathtt{ttc}(V, S') \cup \ldots \\
&= \ldots \cup (\mathtt{ttc}(V, S) \cup \{C\}) \cup \ldots \\
&= (\ldots \cup \mathtt{ttc}(V, S) \cup \ldots) \cup \{C\} \\
&= \Psi(T) \cup \{C\} \\
&\overset{IH}{=} K \cup \{C\} \\
&= K'
\end{aligned}$$

Dabei ist γ eine allquantifizierte Formel, und γ_1 entsteht entsprechend der Zuordnung aus Abbildung 2.2. $K \cup \{\{\gamma_1\}\}$ ist entsprechend Definition 2.16 eine erweiterete Klauselmenge von K (Fall 2) und K' somit eine Klauselmenge mit Verzweigungsliteralen für F.

Fall 3: T' entsteht durch Abschluß eines Asts.

T' entsteht aus T, wie in Fall (4) von Definition 2.9 beschrieben. Dabei wird in T der Knoten $S = \langle W_1, U_1 \rangle$ durch $S' = \langle W_1 \cup \{\gamma_1\}, U_1 \rangle$ ersetzt. Für $\texttt{ttc}(V, S')$ wird berechnet:

$$\begin{aligned}
\texttt{ttc}(V, S') &= \texttt{ttc}(V, \langle W_1 \cup \{\bot\}, U_1 \rangle) \\
&= \texttt{kl}(V, W_1 \cup \{\bot\}) \cup \texttt{lö}(\ldots) \\
&= \{V\} \cup \texttt{kl}(V, W_1) \cup \texttt{lö}(\ldots) \\
&= \{V\} \cup \texttt{ttc}(V, \langle W_1, U_1 \rangle) \\
&= \{V\} \cup \texttt{ttc}(V, S)
\end{aligned}$$

Falls $U_1 = \emptyset$, dann fällt auch hier in Zeile 2 und 3 die Berechnung von $\texttt{lö}(\ldots)$ weg. $\overline{V}$ ist wieder die Menge der Verzweigungsliterale auf dem Pfad von der Wurzel nach S'.

Damit ergibt sich für $\Psi(T')$:

$$\begin{aligned}
\Psi(T') &= \texttt{ttc}(\emptyset, T') \\
&= \ldots \cup \texttt{ttc}(V, S') \cup \ldots \\
&= \ldots \cup (\texttt{ttc}(V, S) \cup \{V\}) \cup \ldots \\
&= (\ldots \cup \texttt{ttc}(V, S) \cup \ldots) \cup \{V\} \\
&= \Psi(T\sigma) \cup \{V\} \\
&\stackrel{*}{=} \Psi(T)\sigma \cup \{V\} \\
&\stackrel{IH}{=} K\sigma \cup \{V\} \\
&= K'
\end{aligned}$$

Die Substitution σ ändert nichts an der Struktur von T, so daß $\Psi(T\sigma) = \Psi(T)\sigma$ gilt (Schritt *). Bei der Transformation wird aus $\bot$ die Klausel V, die die zu den Verzweigungsliteralen des Pfads nach S' komplementären Literale enthält. Das Literal $f \in W_1$ wird zur Klausel $C_1 = \{f\} \cup V$. Da f' auf dem Pfad nach S vorkommt, enthält die Klausel, in die f' übersetzt wird, nur eine Teilmenge der Verzweigungsliterale von C_1. Die Formel f' wird also zu $C_2 = \{f'\} \cup V'$, mit $V' \subseteq V$. Die Literale f und f' werden wie in Definition 2.9, Fall 4 verwendet.

K' ist eine erweiterte Klauselmenge von K (Fall 1 in Definition 2.16), da V als Resolvente von C_1 und C_2 gebildet werden kann. Damit ist K' auch eine Klauselmenge mit Verzweigungsliteralen für F. ■

4.2.2 Korrektheit von Φ

Nun soll auch für die Transformation Φ, die aus einer Klauselmenge und einer Verzweigungsordnung Tableaus erzeugt, gezeigt werden, daß die gebildeten Tableaus der Definition für Tableaus mit Verzweigungsliteralen entsprechen. Als Einschränkung werden zunächst nur totale Verzweigungsordnungen zugelassen.

Satz 4.11
Seien F eine Menge von Tableauformeln, K eine Klauselmenge mit Verzweigungsliteralen für F und $>_K$ eine totale Verzweigungsordnung auf K. Dann ist $T = \Phi(K, >_K)$ ein Tableau mit Verzweigungsliteralen für F.

Beweis:
Es ist zu zeigen, daß für eine Klauselmenge K und eine totale Verzweigungsordnung $>_K$ auf K gilt, daß $T = \Phi(K, >_K)$ ein Tableau mit Verzweigungsliteralen für F ist. Da $>_K$ eine totale Ordnung ist, entsteht bei der Berechnung von $\Phi(K, >_K)$ ein eindeutiges Tableau (Satz 4.2). Auf die Schreibweise mit Klassen von Tableaus wird im folgenden also verzichtet. Der Beweis wird mit Induktion über den Aufbau der Klauselmenge K geführt.

Zunächst wird der Fall $F = \emptyset$ betrachtet. $K = \emptyset$ ist die einzige Klauselmenge mit Verzweigungsliteralen für F (Definition 2.17, Fall 1). Die Ordnung ist in diesem Fall die leere Ordnung. Es gilt:

$$\begin{aligned} T &= \Phi(K, >_K) \\ &= \mathtt{ctt}(\emptyset, >_K) \\ &= \langle \emptyset, \emptyset \rangle \end{aligned}$$

Nach Fall 1 von Definition 2.9 ist $T = \langle \emptyset, \emptyset \rangle$ ein Tableau mit Verzweigungsliteralen für $F = \emptyset$.

Nun wird für $F = \{f_1, \ldots, f_n\}$ der Induktionsbeweis über den Aufbau von K geführt.

Induktionsanfang:

$K = \{C_1, \ldots, C_n\}$ entsteht durch die triviale Transformation aus Abschnitt 2.5.3 aus $F = \{f_1, \ldots, f_n\}$. K ist eine Klauselmenge mit Verzweigungsliteralen für F. Auch in diesem Fall ist $>_K$ die leere Ordnung, da K keine Verzweigungsliterale

enthält. Damit ergibt sich für T:

$$\begin{aligned} T &= \Phi(K, >_K) \\ &= \mathtt{fe}(C_1, >_K, \Phi(K\backslash\{C_1\}, >_K)) \\ &\vdots \\ &= \mathtt{fe}(C_1, >_K, \ldots \Phi(\emptyset, >_K) \ldots) \\ &= \langle\{\mathtt{fma}(C_1), \ldots, \mathtt{fma}(C_n)\}, \emptyset\rangle \\ &= \langle\{f_1, \ldots, f_n\}, \emptyset\rangle \end{aligned}$$

$T = \langle\{f_1, \ldots, f_n\}, \emptyset\rangle$ ist ein Tableau mit Verzweigungsliteralen für F (Definition 2.9, Fall 1).

Induktionsschritt: $K \to K'$

Als Induktionshypothese (IH) gilt $\Phi(K, >_K) = T$, wobei K eine Klauselmenge und T ein Tableau mit Verzweigungsliteralen für F sind. Nun sind drei Fälle zu unterscheiden, die durch die Definition für erweiterte Klauselmengen bestimmt werden (Definition 2.16). K sei die Klauselmenge $\{C_1, \ldots, C_n\}$.

Fall 1:

$K' = \{C_1, \ldots, C_n, D, E\}$ ist eine erweiterte Klauselmenge von K (Definition 2.16, Fall 3). D und E sind Verzweigungsklauseln einer Klausel $C \in K$, und j ist ein neues Verzweigungsliteral, das nur in D und E vorkommt. Die Verzweigungsordnung $>_{K'}$ entsteht durch Erweiterung von $>_K$ um $h >_K \rho(j)$ für alle Verzweigungsatome $h \in K$.

T' wird auf folgende Weise berechnet:

$$\begin{aligned} T' &= \Phi(K', >_{K'}) \\ &= \mathtt{ctt}(K', >_{K'}) \\ &= \mathtt{ctt}(\{C_1, \ldots, C_n, D, E\}, >_{K'}) \\ &= \mathtt{fe}(D, >_{K'}, \mathtt{fe}(E, >_{K'}, \mathtt{ctt}(\{C_1, \ldots, C_n\}, >_{K'}))) \\ &= \mathtt{fe}(D, >_{K'}, \mathtt{fe}(E, >_{K'}, \Phi(K, >_{K'}))) \\ &\overset{*}{=} \mathtt{fe}(D, >_{K'}, \mathtt{fe}(E, >_{K'}, \Phi(K, >_K))) \\ &\overset{IH}{=} \mathtt{fe}(D, >_{K'}, \mathtt{fe}(E, >_{K'}, T)) \end{aligned}$$

Der mit * gekennzeichnete Schritt ist erlaubt, da j und $\overline{j}$ nur in D und E vorkommen. D und E sind jedoch nicht in K enthalten, so daß die Verwendung von

$>_K$ gegenüber von $>_{K'}$ auf der Menge K keinen Unterschied erzeugt.

Die Funktion `fe` fügt die Klauseln D und E an allen Blättern B_i im Teiltableau unterhalb des Knotens, in den die Klausel C eingefügt wurde, ein, da D und E genau die Verzweigungsliterale aus C enthalten. Außerdem kommen noch die Verzweigungsliterale j oder $\overline{j}$ in D und E vor (Definition 2.16). Da $\rho(j)$ bezüglich $>_{K'}$ das einzige Minimum ist, wird T durch die Funktion `fe` an allen Blättern B_i um eine mit j gekennzeichnete Verzweigung erweitert. Das dabei erhaltene Tableau T' ist nach Fall 3 von Definition 2.9 ein Tableau mit Verzweigungsliteralen.

Fall 2:

$K' = \{C_1, \ldots, C_n, D\}$ ist eine erweiterte Klauselmenge von K (Definition 2.16, Fall 2). Dabei entsteht D durch Variablenumbenennung einer Klausel $E \in K$ aus der universellen Teilmenge von K. Die Verzweigungsordnung $>_{K'}$ stimmt mit $>_K$ überein. T' wird wie folgt berechnet:

$$\begin{aligned}
T' &= \Phi(K', >_{K'}) \\
&= \mathtt{ctt}(K', >_{K'}) \\
&= \mathtt{ctt}(\{C_1, \ldots, C_n, D\}, >_{K'}) \\
&= \mathtt{fe}(D, >_{K'}, \mathtt{ctt}(\{C_1, \ldots, C_n\}, >_{K'})) \\
&= \mathtt{fe}(D, >_{K'}, \Phi(K, >_{K'})) \\
&= \mathtt{fe}(D, >_{K'}, \Phi(K, >_K)) \\
&\overset{IH}{=} \mathtt{fe}(D, >_{K'}, T)
\end{aligned}$$

Die Klausel $E \in K$ ist im Tableau T eine allquantifizierte Formel. Sie steht im Multiknoten W_1, dessen Stelle im Tableau durch die Verzweigungsliterale in E bestimmt wird. D entsteht durch Variablenumbenennung aus E und ist Element der starren Teilmenge von K'. D enthält dieselben Verzweigungsliterale wie E. Dadurch wird D durch die Funktion `fe` in den gleichen Multiknoten W_1 eingefügt, in dem auch E vorkommt. Da T ein Tableau mit Verzweigungsliteralen für F ist, ist das um D erweiterte Tableau nach Fall 2 von Definition 2.9 auch ein Tableau mit Verzweigungsliteralen für F.

Fall 3:

In diesem Fall gilt $K' = K\sigma \cup \{D\}$. K' ist eine erweiterte Klauselmenge von K (Definition 2.16, Fall 1). Dabei ist D eine Resolvente von $E \in K$ und $F \in K$. Für E und F soll $|Fl(E)| = |Fl(F)| = 1$ gelten. E und F enthalten also jeweils nur

ein Formelliteral. D entsteht durch Resolventenbildung an diesem Formelliteral. Die dabei entstandene Substitution σ wird auf K angewendet. Außerdem dürfen E und F keine komplementären Verzweigungsliterale enthalten. Sonst enthält auch D komplementäre Literale und ist eine Tautologie.

Die Verzweigungsordnung $>_{K'}$ enthält gegenüber $>_K$ keine neuen Verzweigungsliterale. Es muß lediglich die Substitution σ auf die Verzweigungsliterale der Ordnung angewendet werden.

Für T' ergibt sich:

$$\begin{aligned} T' &= \Phi(K', >_{K'}) \\ &= \mathtt{ctt}(K\sigma \cup \{D\}, >_{K'}) \\ &= \mathtt{fe}(D, >_{K'}, \mathtt{ctt}(K\sigma, >_{K'})) \\ &= \mathtt{fe}(D, >_{K'}, \Phi(K\sigma, >_{K'})) \\ &= \mathtt{fe}(D, >_{K'}, \Phi(K\sigma, >_{K\sigma})) \\ &\overset{*}{=} \mathtt{fe}(D, >_{K'}, \Phi(K, >_K)\sigma) \\ &\overset{IH}{=} \mathtt{fe}(D, >_{K'}, T\sigma) \end{aligned}$$

Der mit * gekennzeichnete Schritt kann durchgeführt werden, da die Anwendung der Substitution keine Änderung der Struktur des Tableaus erzeugt.

Die zu der Klausel F korrespondierende Tableauformel kommt in T im Knoten S vor und die zu E korrespondierende in einem Knoten auf dem Pfad von der Wurzel nach S. Außerdem sind diese beiden Formeln unter der Substitution σ komplementäre Literale. Die Klausel D enthält nur Verzweigungsliterale. Durch `fe` wird das Abschlußsymbol $\perp$ im Knoten S eingefügt. Das so entstandene Tableau T' ist nach Fall 4 von Definition 2.9 ein Tableau mit Verzweigungsliteralen für F.

Falls die Klauseln E und F mehrere Formelliterale enthalten, läßt sich nur mit mehreren Tableauerweiterungsschritten zeigen, wie T' aus T entsteht. Es wird eine β-Regel-Erweiterung mit anschließendem Abschluß eines der beiden Äste durchgeführt. Das resultierende Tableau ist also auch ein Tableau mit Verzweigungsliteralen. Der genaue Ablauf wird in Abschnitt 6.2.1 verdeutlicht. ■

Aus der Gültigkeit des letzten Satzes 4.11 folgt sofort die Erweiterung auf beliebige Verzweigungsordnungen. Es gilt:

Korollar 4.12
Sei F eine Menge von Tableauformeln, K eine Klauselmenge mit Verzweigungsliteralen für F und $>_K$ eine beliebige Verzweigungsordnung auf K. Dann enthält die Klasse $[T] = \Phi(K, >_K)$ nur Tableaus mit Verzweigungsliteralen für F.

Da Vergleiche zwischen ungeordneten Literalen bei der weiteren Bearbeitung durch Φ beibehalten werden müssen (Definition von `fe` in Abschnitt 3.3), können alle Tableaus aus $[T]$ unter Verwendung von totalen Ordnungen aus $\tau(>_K)$ berechnet werden. Diese totalen Ordnungen lassen sich aus der beliebigen durch Ordnen der nicht geordneten Verzweigungsatome bilden.

4.3 Umkehrbarkeit der Transformationen

Nachdem im letzten Abschnitt gezeigt wurde, daß die Transformationen wirklich die gewünschten Ergebnisse liefern, soll nun untersucht werden, ob die Anwendung der Verknüpfung der Transformationen Φ und Ψ wieder das ursprüngliche Tableau, auf das sie angewendet wurde, ergibt.

4.3.1 Umkehrtransformation von Ψ

Mit der Transformation Φ kann aus der durch Ψ für ein Tableau T erzeugten Klauselmenge wieder T gebildet werden, wie der nächste Satz beziehungsweise dessen Beweis zeigt.

Satz 4.13
Sei T ein Tableau mit Verzweigungsliteralen für die Tableauformelmenge F. Es gilt $\Phi(\Psi(T)) = T$.

Beweis:
Es ist zu zeigen, daß die Anwendung der Transformation Φ auf die Klauselmenge $\Psi(T)$ wieder das Tableau T ergibt. Dafür muß ein Induktionsbeweis über den Aufbau des Tableaus geführt werden.

Für den Fall $F = \emptyset$ gibt es nur ein mögliches Tableau mit Verzweigungsliteralen für F. Dieses ist $T = \langle \emptyset, \emptyset \rangle$. Es gilt:

$$\begin{aligned}
\Phi(\Psi(T)) &= \Phi(\Psi(\langle \emptyset, \emptyset \rangle)) \\
&= \Phi(\mathtt{ttc}(\emptyset, \langle \emptyset, \emptyset \rangle)) \\
&= \Phi(\mathtt{kl}(\emptyset, \emptyset), >_T) \\
&= \Phi(\emptyset, >_T) \\
&= \mathtt{ctt}(\emptyset, >_T) \\
&= \langle \emptyset, \emptyset \rangle \\
&= T
\end{aligned}$$

Nun werden der Fall $F = \{f_1, \ldots, f_n\}$ betrachtet und der Induktionsbeweis über den Aufbau von T geführt:

Induktionsanfang:

Hier besteht $T = \langle \{f_1, \ldots, f_n\}, \emptyset \rangle$ nur aus einem Multiknoten und besitzt keine Unterbäume (Fall 1 von Definition 2.9). Der Multiknoten enthält also alle Formeln aus F. Es ergibt sich:

$$\begin{aligned}
\Phi(\Psi(T)) &= \Phi(\Psi(\langle \{f_1, \ldots, f_n\}, \emptyset \rangle)) \\
&= \Phi(\mathtt{ttc}(\emptyset, \langle \{f_1, \ldots, f_n\}, \emptyset \rangle)) \\
&= \Phi(\mathtt{kl}(\emptyset, \{f_1, \ldots, f_n\}), >_T) \\
&= \Phi(\{C_1, \ldots, C_n\}, >_T) \\
&= \mathtt{fe}(C_1, >_T, \Phi(K \backslash \{C_1\}, >_T)) \\
&\vdots \\
&= \mathtt{fe}(C_1, >_T, \ldots \Phi(\emptyset, >_T) \ldots) \\
&= \langle \{\mathtt{fma}(C_1), \ldots, \mathtt{fma}(C_n)\}, \emptyset \rangle \\
&= \langle \{f_1, \ldots, f_n\}, \emptyset \rangle \\
&= T
\end{aligned}$$

Dabei ist C_i eine Klausel mit Verzweigungsliteralen für f_i für $1 \leq i \leq n$. Die durch Ψ gebildete Klauselmenge $\{C_1, \ldots, C_n\}$ enthält keine Verzweigungsliterale, und die Ordnung $>_T$ ist die leere Ordnung, da das Tableau T keine Unterbäume besitzt. Durch die Anwendung von Φ entsteht wieder das Ausgangstableau $T = \langle \{f_1, \ldots, f_n\}, \emptyset \rangle$.

Induktionsschritt: $T \rightarrow T'$

T' entsteht durch Erweiterung des Tableaus T. Die drei Fälle aus Definition 2.9 müssen getrennt betrachtet werden. Als Induktionshypothese (IH) gilt

$$\Phi(\Psi(T)) = T,$$

wobei T ein Tableau mit Verzweigungsliteralen für F ist.

Fall 1: T' entsteht durch eine β-Regel-Erweiterung.

In T werden alle Blätter $B_i = \langle W_i, \emptyset \rangle$ des Teilbaums mit der Wurzel M, in der die β-Formel vorkommt, durch

$$B_i' = \langle W_i, ((j, \langle \{\beta_1\}, \emptyset \rangle), (\overline{j}, \langle \{\beta_2\}, \emptyset \rangle)) \rangle$$

ersetzt (Fall 3 von Definition 2.9). Das Ergebnis ist das Tableau T'. Wie schon im Beweis von Satz 4.10 (Induktionsschritt, Fall 1) gezeigt wurde, ergibt sich für $\Psi(T')$:

$$\Psi(T') = \Psi(T) \cup \{D, E\}$$

Dabei werden abkürzend $D = \{V \cup \{\overline{j}\} \cup \{\beta_1\}\}$ und $E = \{V \cup \{j\} \cup \{\beta_2\}\}$ verwendet. $\overline{V}$ ist die Menge der Verzweigungsliterale auf dem Pfad von der Wurzel in T nach M, und j ist ein neues Verzweigungsliteral, das bei der Erweiterung von T eingeführt wurde.

Es gilt folgende Berechnung:

$$\begin{aligned}
\Phi(\Psi(T')) &= \Phi(\mathtt{ttc}(\emptyset, T')) \\
&= \Phi(\Psi(T) \cup \{D, E\}) \\
&= \mathtt{ctt}(\Psi(T) \cup \{D, E\}, >_{T'}) \\
&= \mathtt{fe}(D, >_{T'}, \mathtt{fe}(E, >_{T'}, \mathtt{ctt}(\Psi(T), >_{T'}))) \\
&\overset{*}{=} \mathtt{fe}(D, >_{T'}, \mathtt{fe}(E, >_{T'}, \Phi(\Psi(T)))) \\
&\overset{IH}{=} \mathtt{fe}(D, >_{T'}, \mathtt{fe}(E, >_{T'}, T)) \\
&\overset{**}{=} T'
\end{aligned}$$

Bei der Bearbeitung von $\Psi(T')$ wird auch die tableaueindeutige Verzweigungsordnung $>_{T'}$ erzeugt, nach der dann durch die Funktion Φ wieder das Tableau T' hergestellt wird. Das bei T' neu hinzugekommene Verzweigungsliteral j ist das einzige minimale Element in $>_{T'}$ und kommt in T nicht vor. Die Ordnung $>_{T'}$ kann also auch zur Erzeugung des Tableaus T verwendet werden (Schritt *).

Da die Klauseln D und E nur die Verzweigungsliterale aus V und noch j oder $\overline{j}$ enthalten, wird durch die Funktion `fe` jedes Blatt B_i im Teilbaum unter M um eine mit j markierte Verzweigung erweitert (Schritt **). Das so entstandene Tableau ist wieder das Ausgangstableau.

Fall 2: T' entsteht durch eine γ-Regel-Erweiterung.

Dieser Teil des Induktionsschritts verläuft weitgehend analog zu Fall 1. T' entsteht aus T durch Ersetzen eines Knotens $S = \langle W_1, U_1 \rangle$ durch $S' = \langle W_1 \cup \{\gamma_1\}, U_1 \rangle$ (Fall 2 von Definition 2.9). Im Beweis von Satz 4.10 (Induktionsschritt, Fall 2) wurde schon gezeigt, daß für $\Psi(T')$ gilt:

$$\Psi(T') = \Psi(T) \cup \{C\}$$

Dabei ist $C = \{V \cup \{\gamma_1\}\}$ die Klausel, die aus der neu in das Tableau T eingefügten Formel entstanden ist. Es gilt:

$$\begin{aligned}
\Phi(\Psi(T')) &= \Phi(\mathtt{ttc}(\emptyset, T')) \\
&= \Phi(\Psi(T) \cup \{C\}) \\
&= \mathtt{ctt}(\Psi(T) \cup \{C\}, >_{T'}) \\
&= \mathtt{fe}(C, >_{T'}, \mathtt{ctt}(\Psi(T), >_{T'})) \\
&\overset{*}{=} \mathtt{fe}(C, >_{T'}, \Phi(\Psi(T))) \\
&\overset{IH}{=} \mathtt{fe}(C, >_{T'}, T) \\
&\overset{**}{=} T'
\end{aligned}$$

Auch in diesem Fall ist $>_{T'}$ die Ordnung, die bei der Bearbeitung von $\Psi(T')$ entsteht. Da T' die gleichen Verzweigungsliterale wie T enthält, kann $>_{T'}$ auch zum Aufbau von T verwendet werden (Schritt *). C enthält die Verzweigungsliterale des Pfads von der Wurzel in T nach S, nämlich die aus V. Dadurch wird C durch die Funktion `fe` in den Knoten S eingefügt, und es entsteht wieder das Tableau T' (Schritt **).

Fall 3: T' entsteht durch Abschluß eines Asts.

In diesem Fall entsteht T' aus T durch den Abschluß eines Asts. In den Knoten $S = \langle W_1, U_1 \rangle$ aus T wird das Abschlußsymbol $\perp$ eingefügt. Außerdem wird die Substitution σ auf T angewendet. σ ist die Substitution, unter der zwei Literale auf dem Pfad nach S komplementär werden. Es gilt

$$\Psi(T') = \Psi(T)\sigma \cup \{V\},$$

wie schon im Beweis von Satz 4.10 (Induktionsschritt, Fall 3) gezeigt wurde. Damit entsteht folgende Berechnung:

$$\begin{aligned}
\Phi(\Psi(T')) &= \Phi(\mathtt{ttc}(\emptyset, T')) \\
&= \Phi(\Psi(T)\sigma \cup \{V\}, >_{T'}) \\
&= \mathtt{ctt}(\Psi(T)\sigma \cup \{V\}, >_{T'}) \\
&= \mathtt{fe}(V, >_{T'}, \mathtt{ctt}(\Psi(T)\sigma, >_{T'})) \\
&\overset{*}{=} \mathtt{fe}(V, >_{T'}, \mathtt{ctt}(\Psi(T), >_T)\sigma) \\
&= \mathtt{fe}(V, >_{T'}, \Phi(\Psi(T))\sigma) \\
&\overset{IH}{=} \mathtt{fe}(V, >_{T'}, T\sigma) \\
&= T'
\end{aligned}$$

Durch $\Psi(T')$ wird wieder die Verzweigungsordnung $>_{T'}$ berechnet. $>_T$ ist die Ordnung, aus der $>_{T'}$ durch Anwendung der Substitution σ entsteht. Wendet man σ auf das unter $>_T$ aus der Klauselmenge $\Psi(T')$ entstandene Tableau an, erhält man das gleiche Tableau, das unter $>_{T'}$ aus $\Psi(T)\sigma$ aufgebaut wird (Schritt *). Durch das Einfügen der Klausel $\{V\}$ in $T\sigma$ wird der Knoten S um das Abschlußsymbol erweitert. Man erhält also wieder T'. ■

4.3.2 Umkehrtransformation von Φ

Die Verknüpfung von Φ und Ψ in umgekehrter Reihenfolge ist schwieriger zu behandeln, da neben der Klauselmenge noch die Ordnung berücksichtigt werden muß. Es werden zunächst nur totale Verzweigungsordnungen verwendet, so daß mit der Transformation Φ aus einer Klauselmenge K und einer dazugehörenden totalen Ordnung $>_K$ nur ein Tableau gebildet werden kann. Es soll gezeigt werden, daß die Transformation Ψ dieses Tableau wieder in die Klauselmenge K und die Ordnung $>_T$ zurückwandelt. Die Wiederherstellung der Klauselmenge ist dabei die leichtere Aufgabe. Schwieriger sind die Ordnungen.

Es werden auch hier nur die Verzweigungsordnungen $>_K$ verwendet, die die Verzweigungsatome aus K vergleichen und keine weiteren, nicht in K vorkommenden Verzweigungsatome. Vergleiche mit Verzweigungsatomen, die nicht in der Klauselmenge enthalten sind, können aus dem Tableau nicht wieder hergestellt werden, da diese Verzweigungsatome nach der Transformation Φ auch im Tableau nicht enthalten sind.

Die durch Ψ aus dem Tableau $\Phi(K, >_K)$ erzeugte Ordnung $>_T$ ist, wie schon in Abschnitt 4.1 gezeigt wurde, nicht unbedingt total auf K. Sie ist aber in jedem

Fall tableaueindeutig. Falls sie dennoch total ist, dann sind $>_K$ und $>_T$ gleich. Wenn $>_T$ nur tableaueindeutig ist, so gilt $>_K \in \tau(>_T)$. Nun wird der folgende Satz gezeigt:

Satz 4.14
Seien K eine Klauselmenge mit Verzweigungsliteralen für die Formelmenge F und $>_K$ eine totale Verzweigungsordnung auf K. Es gilt $\Psi(\Phi(K, >_K)) = \langle K, >_T \rangle$ mit $>_K = >_T$ oder $>_K \in \tau(>_T)$.

Beweis:
Es ist zu zeigen, daß durch die Anwendung von Ψ auf das Tableau $\Phi(K, >_K)$ wieder die Klauselmenge K und die Verzweigungsordnung $>_K$ entstehen. Hierfür muß ein Induktionsbeweis über den Aufbau von K geführt werden.

Zunächst wird der Sonderfall $F = \emptyset$ betrachtet. Nach Definition 2.17, Fall 1 ist $K = \emptyset$ die einzige Klauselmenge mit Verzweigungsliteralen für F, und $>_K$ ist die leere Ordnung. Es gilt:

$$\begin{aligned}
\Psi(\Phi(K, >_K)) &= \Psi(\Phi(\emptyset, >_K)) \\
&= \Psi(\mathtt{ctt}(\emptyset, >_K)) \\
&= \Psi(\langle \emptyset, \emptyset \rangle) \\
&= \mathtt{ttc}(\emptyset, \langle \emptyset, \emptyset \rangle) \\
&= \mathtt{kl}(\emptyset, \emptyset) \\
&= \langle \emptyset, >_T \rangle \\
&\stackrel{*}{=} \langle K, >_K \rangle
\end{aligned}$$

Mit der Transformation Φ wird das Tableau $\langle \emptyset, \emptyset \rangle$ erzeugt, aus dem mit Ψ wieder die leere Klauselmenge gebildet wird. Ψ erzeugt außerdem die Verzweigungsordnung $>_T$, die in diesem Fall die leere Ordnung ist. Es gilt also $>_T = >_K$ (Schritt *).

Nun wird für $F = \{f_1, \ldots, f_n\}$ der Induktionsbeweis über den Aufbau von K geführt.

Induktionsanfang:

Die Klauselmenge $K = \{C_1, \ldots, C_n\}$ entsteht durch die triviale Transformation aus Abschnitt 2.5.3 aus $F = \{f_1, \ldots, f_n\}$. Da K keine Verzweigungsliterale

enthält, ist $>_K$ wieder die leere Ordnung. Es gilt:

$$\begin{aligned}
\Psi(\Phi(K,>_K)) &= \Psi(\Phi(\{C_1,\ldots,C_n\},>_K)) \\
&= \Psi(\mathtt{fe}(C_1,>_K,\Phi(\{C_2,\ldots,C_n\},>_K))) \\
&\vdots \\
&= \Psi(\mathtt{fe}(C_1,>_K,\ldots\Phi(\emptyset,>_K)\ldots)) \\
&= \Psi((\{\mathtt{fma}(C_1),\ldots,\mathtt{fma}(C_n)\},\emptyset)) \\
&= \Psi(\langle\{f_1,\ldots,f_n\},\emptyset\rangle) \\
&= \mathtt{ttc}(\emptyset,\langle\{f_1,\ldots,f_n\},\emptyset\rangle) \\
&= \langle\mathtt{kl}(\emptyset,\{f_1,\ldots,f_n\}),>_T\rangle \\
&= \langle\{C_1,\ldots,C_n\},>_T\rangle \\
&= \langle K,>_K\rangle
\end{aligned}$$

C_i und f_i (für $1 \leq i \leq n$) sind jeweils die miteinander korrespondierende Klausel und Tableauformel. Durch Φ wird das Tableau $\langle\{f_1,\ldots,f_n\},\emptyset\rangle$ erzeugt, bei dem alle Formeln im Multiknoten an der Wurzel enthalten sind und das keine Unterbäume besitzt. Daraus wird mit Ψ wieder die Klauselmenge K hergestellt. Weiterhin erhält man durch Ψ die leere Ordnung $>_T$. Es gilt also $>_K = >_T$.

Induktionsschritt: $K \rightarrow K'$

K' ist eine erweiterte Klauselmenge von $K = \{C_1,\ldots,C_n\}$. Es müssen also die Fälle aus Definition 2.16 unterschieden werden.

Die Induktionshypothese (IH) ist $\Psi(\Phi(K,>_K)) = \langle K,>_T\rangle$, wobei K eine Klauselmenge mit Verzweigungsliteralen und $>_K$ und $>_T$ Verzweigungsordnungen für K sind. Es gilt $>_K = >_T$ oder $>_K \in \tau(>_T)$.

Fall 1:

Nach Definition 2.16, Fall 3 ist $K' = \{C_1,\ldots,C_n,D,E\}$ eine erweiterte Klauselmenge von K . D und E sind Verzweigungsklauseln einer Klausel $C \in K$, und j ist ein neues Verzweigungsliteral, das nicht in K vorkommt. Es ist also nur in D und E enthalten. Die Verzweigungsordnung $>_{K'}$ entsteht durch Erweiterung von $>_K$ um $h>_K\rho(j)$ für alle Verzweigungsatome $h \in K$. Wie schon im Beweis von Satz 4.11 (Induktionsschritt, Fall 1) gezeigt wurde, gilt:

$$\Phi(K',>_{K'}) = \mathtt{fe}(D,>_{K'},\mathtt{fe}(E,>_{K'},\Phi(K,>_K)))$$

Es wird folgendes berechnet:

$$\begin{aligned}
\Psi(\Phi(K', >_{K'})) &= \Psi(\Phi(\{C_1, \ldots, C_n, D, E\}, >_{K'})) \\
&= \Psi(\mathtt{fe}(D, >_{K'}, \mathtt{fe}(E, >_{K'}, \Phi(K, >_K)))) \\
&= \mathtt{ttc}(\emptyset, \mathtt{fe}(D, >_{K'}, \mathtt{fe}(E, >_{K'}, \Phi(K, >_K)))) \\
&\stackrel{*}{=} \ldots \cup \mathtt{ttc}(V_i, B_i') \cup \ldots \\
&= \ldots \cup (\mathtt{ttc}(V_i, B_i) \cup \{D_i, E_i\}) \cup \ldots \\
&= (\ldots \cup \mathtt{ttc}(V_i, B_i) \cup \ldots) \cup \{D, E\} \\
&\stackrel{**}{=} \langle \Psi(\Phi(K, >_K)) \cup \{D, E\}, >_{T'} \rangle \\
&\stackrel{IH}{=} \langle K \cup \{D, E\}, >_{T'} \rangle \\
&= \langle K', >_{T'} \rangle
\end{aligned}$$

Aus dem um D und E erweiterten Tableau $\Phi(K, >_K)$ wird durch die Funktion `ttc` wieder die um D und E erweiterte Klauselmenge hergestellt (Schritt * bis **, analog zum Beweis von Satz 4.10).

Da die Erzeugung der Klauselmenge und der Ordnung durch die Transformation Ψ bei deren Definition in Abschnitt 3.4 getrennt wurde, ist die Bildung der Ordnung $>_{T'}$ in der obige Berechnung nur ungenau wiedergegeben. Durch Ψ wird aus $\Phi(K, >_K)$ die Ordnung $>_T$ berechnet (IH), die dann bei der Erzeugung der Klauseln D und E (Schritt * bis **) um die Vergleiche $\rho(h) >_{T'} \rho(j)$ erweitert wird. Und zwar für alle Literale h auf dem Pfad zum Multiknoten M, der die zu C korrespondierende Formel enthält, und für alle h, die im Teilbaum unter M vorkommen. Diese Vergleiche werden mit der Funktion `ord` (Abschnitt 3.4) gebildet.

Die Ordnung $>_{K'}$ kann auch noch Vergleiche für j enthalten, die nicht in $>_{T'}$ vorkommen. Die Verzweigungsordnung $>_{T'}$ ist jedoch auch ohne diese Vergleiche tableaueindeutig, da C und D nur in Teilbäumen vorkommen, in denen die Verzweigungsliterale, mit denen j durch $>_{K'}$ noch verglichen wird, nicht vorkommen. Es gilt also $>_{K'} \in \tau(>_{T'})$.

Fall 2:

In diesem Fall ist $K' = \{C_1, \ldots, C_n, D\}$ eine erweiterte Klauselmenge von K (Definition 2.16, Fall 2). Dabei entsteht D durch Variablenumbenennung einer Klausel $E \in K$ aus der universellen Teilmenge von K. Die Verzweigungsordnung $>_{K'}$ stimmt mit $>_K$ überein. Im Beweis von Satz 4.11 (Induktionsschritt, Fall 2)

wurde schon gezeigt, daß für $\Phi(K', >_{K'})$ gilt:

$$\Phi(K', >_{K'}) = \mathtt{fe}(D, >_{K'}, \Phi(K, >_K))$$

Damit wird berechnet:

$$\begin{aligned}
\Psi(\Phi(K', >_{K'})) &= \Psi(\Phi(\{C_1, \ldots, C_n, D\}, >_{K'})) \\
&= \Psi(\mathtt{fe}(D, >_{K'}, \Phi(K, >_K))) \\
&= \mathtt{ttc}(\emptyset, \mathtt{fe}(D, >_{K'}, \Phi(K, >_K))) \\
&= \ldots \cup \mathtt{ttc}(V, S') \cup \ldots \\
&= \ldots \cup (\mathtt{ttc}(V, S) \cup \{D\}) \cup \ldots \\
&= (\ldots \cup \mathtt{ttc}(V, B) \cup \ldots) \cup \{D\} \\
&= \langle \Psi(\Phi(K, >_K)) \cup \{D\}, >_{T'} \rangle \\
&\overset{IH}{=} \langle K \cup \{D\}, >_{T'} \rangle \\
&= \langle K', >_{T'} \rangle
\end{aligned}$$

Auch in diesem Fall wird durch die Funktion Ψ aus dem Tableau $\Phi(K, >_K)$ die Ordnung $>_T$ hergestellt (IH). Da durch das Einfügen von D keine Änderung der Struktur des Tableaus vorgenommen wird, ist $>_{T'}$ die gleiche Ordnung wie $>_T$. Für die durch Ψ erzeugte Ordnung $>_{T'}$ gilt also, daß $>_{K'} = >_{T'}$ oder $>_{K'} \in \tau(>_{T'})$ je nachdem, ob $>_K = >_T$ oder $>_K \in \tau(>_T)$.

Fall 3:

In diesem Fall ist $K' = K\sigma \cup \{D\}$ die erweiterte Klauselmenge von K (Definition 2.16, Fall 1). Dabei ist D eine Resolvente von $E \in K$ und $F \in K$. D entsteht durch Resolventenbildung an einem Formelliteral aus D und E. Die dabei entstandene Substitution σ wird auf K angewendet. Die Klauseln E und F dürfen keine komplementären Verzweigungsliterale enthalten, da D sonst eine Tautologie ist.

Die Verzweigungsordnung $>_{K'}$ enthält gegenüber $>_K$ keine neuen Verzweigungsliterale. Es muß lediglich die Substitution σ auf die Verzweigungsliterale der Ordnung angewendet werden.

Wie schon im Beweis von Satz 4.11 (Induktionsschritt, Fall 3) gezeigt wurde, gilt

$$\Phi(K', >_{K'}) = \mathtt{fe}(D, >_{K'}, \Phi(K, >_K)\sigma),$$

Damit wird berechnet:

$$\begin{aligned}
\Psi(\Phi(K', >_{K'})) &= \Psi(\Phi(K\sigma \cup \{D\}, >_{K'})) \\
&= \Psi(\mathtt{fe}(D, >_{K'}, \Phi(K, >_K)\sigma)) \\
&= \mathtt{ttc}(\emptyset, \mathtt{fe}(D, >_{K'}, \Phi(K, >_K)\sigma)) \\
&= \ldots \cup \mathtt{ttc}(V, S') \cup \ldots \\
&= \ldots \cup (\mathtt{ttc}(V, S) \cup \{D\}) \cup \ldots \\
&= (\ldots \cup \mathtt{ttc}(V, S) \cup \ldots) \cup \{D\} \\
&= \langle \Psi(\Phi(K, >_K)\sigma) \cup \{D\}, >_T \sigma \rangle \\
&\overset{*}{=} \langle \Psi(\Phi(K, >_K))\sigma \cup \{D\}, >_T \sigma \rangle \\
&\overset{IH}{=} \langle K\sigma \cup \{D\}, >_K \sigma \rangle \\
&= \langle K', >_{K'} \rangle
\end{aligned}$$

Schritt * wird analog zum Beweis von Satz 4.10 geführt. Mit Ψ wird die Ordnung $>_{T'}$ durch Anwendung von σ auf $>_T$ gebildet. Da die Ordnungen $>_K$ und $>_T$ in diesem Fall gleich sind, gilt auch $>_{K'} = >_{T'}$. ■

Aus der Gültigkeit des letzten Satzes wird das folgende Korollar geschlossen:

Korollar 4.15
Seien K eine Klauselmenge mit Verzweigungsliteralen für die Formelmenge F und $>_K$ eine beliebige Verzweigungsordnung auf K. Es gilt $\Psi(\Phi(K, >_K)) = \langle K, O \rangle$, wobei $O = \{>_T^1, \ldots, >_T^n\}$ eine Menge von tableaueindeutigen Verzweigungsordnungen ist und $\tau(>_K) \subseteq \bigcup_{i=1}^{n} \tau(>_T^i)$.

Mit Φ kann unter einer beliebige Verzweigungsordnung $>_K$ aus K eine Klasse $[T]$ von Tableaus erzeugt werden. Alle Tableaus dieser Klasse werden mit der Transformation Ψ wieder in die Klauselmenge K zurückgewandelt. Allerdings erzeugt Ψ nur tableaueindeutige Ordnungen, so daß die Herstellung von $>_K$ nicht möglich ist. Statt dessen wird die Menge O von Verzweigungsordnungen erzeugt.

4.4 Erfüllbarkeit

Mit den Sätzen 4.10, 4.11, 4.13 und 4.14 und denen aus den Abschnitten 2.4.3 und 2.5.6 lassen sich die folgenden Aussagen schließen:

Korollar 4.16
Sei T ein Tableau mit Verzweigungsliteralen für die Tableauformelmenge F.

- *Ist T ein erfüllbares Tableau, dann ist $K = \Psi(T)$ eine erfüllbare Klauselmenge für F.*
- *Ist T geschlossen, dann läßt sich aus der Klauselmenge mit Verzweigungsliteralen $K = \Psi(T)$ die leere Klausel ableiten.*

Korollar 4.17
Sei K eine Klauselmenge mit Verzweigungsliteralen für die Formelmenge F.

- *Ist K eine erfüllbare Klauselmenge und $>_K$ eine Verzweigungsordnung auf K, dann ist $T = \Phi(K, >_K)$ ein erfüllbares Tableau für F.*
- *Aus K läßt sich die leere Klausel ableiten, dann gibt es ein geschlossenes Tableau mit Verzweigungsliteralen T für F.*

5 Integer-programming-Probleme

Nachdem die Transformationen zwischen Tableaus und Klauselmengen eingeführt wurden, soll nun eine Transformation von Klauselmengen in Integer-programming-Probleme vorgestellt werden (Abschnitt 5.1). Im darauffolgenden Abschnitt 5.2 wird ein Verfahren angegeben, mit dem die Unerfüllbarkeit einer Klauselmenge K durch Bearbeitung des zu K berechneten 0-1-Integer-Programms festgestellt werden kann. Schließlich wird in Abschnitt 5.3 noch eine Bemerkung zur Bedeutung der bei diesem Verfahren neu erzeugten Integer-programming-constraints gemacht.

5.1 Transformation von Klauselmengen in Integer-Programme

Klauselmengen können in einfacher Weise in Integer-programming-Probleme übertragen werden [Hoo92a, Hoo92b]. Da es sich bei den Klauseln um Disjunktionen von Literalen handelt, wird hier die übliche Übersetzung in Integer-programming-Ungleichungen (constraints[1]) verwendet. Aus einer Klausel $\{p_1, \ldots, p_n, \neg p_{n+1}, \ldots, \neg p_m\}$ wird der Integer-programming-constraint

$$p_1 + \ldots + p_n + (1 - p_{n+1}) + \ldots + (1 - p_m) \geq 1$$

erzeugt. Dabei sind p_i, $1 \leq i \leq m$, prädikatenlogische Atome, wobei $p_1, \ldots, p_n$ die positiven und $\neg p_{n+1}, \ldots, \neg p_m$ die negativen Literale in der Klausel sind. Die einzelnen Atome werden dann als Integer-programming-Variablen verwendet.

Bei der Definition der Transformationsfunktion werden neben den Typen aus Tabelle 3.1 noch zwei weitere verwendet. Und zwar $\mathcal{IP}$ für Integer-Programme und $\mathcal{IPC}$ für Integer-programming-constraints. Die Transformation einer Klauselmenge mit Verzweigungsliteralen in eine Menge von Integer-programming-constraints, also in ein Integer-Programm, wird durch die Funktion Ω beschrieben:

$$\Omega : \mathcal{K} \rightarrow \mathcal{IP}$$

$$\Omega(K) = \begin{cases} \emptyset & falls\ K = \emptyset \\ ipc(C) \cup \Omega(K \backslash \{C\}) & sonst,\ wobei\ C \in K\ beliebig\ ist \end{cases}$$

[1] Im folgenden wird der Ausdruck *constraint* weiter verwendet.

Dabei wird mit der Funktion $ipc(C)$ wie oben beschrieben der zur Klausel C korrespondierende Integer-programming-constraint berechnet.

$$ipc : \mathcal{C} \rightarrow \mathcal{IPC}$$

$$\begin{aligned} ipc(\{p_1, \ldots, p_n, \neg p_{n+1}, \ldots, \neg p_m\}) &= \{p_1 + \ldots + p_n + \\ &\quad (1 - p_{n+1}) + \ldots + (1 - p_m) \geq 1\} \end{aligned}$$

In analoger Weise läßt sich aus einem Integer-Program auch wieder eine Klauselmenge bilden.

5.2 Ein Beweisverfahren mit Integer-Programmen

In [BH96] wird ein Verfahren vorgestellt, das semantische Tableaus mit Hilfe von 0-1-Integer-Programmen auf ihre Unerfüllbarkeit hin überprüft. Dabei werden die 0-1-Integer-Programme nicht dazu benutzt, einzeln zu testen, ob jeder Ast im Tableau geschlossen ist, sondern es wird mit einer speziellen Umformung das gesamte Tableau mit seiner Struktur in ein Integer-Programm übersetzt.

Mit der in Kapitel 3 beschriebenen Transformation von Tableaus in Klauselmengen mit Verzweigungsliteralen und der anschließenden Übersetzung durch Ω wird genau ein solches Integer-Programm für das Tableau T erzeugt. $\Omega(\Psi(T))$ stellt also genau die in [BH96] beschriebene Transformation dar. Die Unerfüllbarkeit des Tableaus kann nun mit dem zugehörigen Integer-Programm und darauf angewendeten Integer-programming-Lösungsverfahren überprüft werden.

Das in [BH96] für Tableaus vorgestellte Verfahren wird hier für Klauselmengen mit Verzweigungsliteralen angegeben. Es wird also die Unerfüllbarkeit einer Klauselmenge mit Hilfe des dazu berechneten Integer-Programms festgestellt:

1. Berechne das zur Klauselmenge mit Verzweigungsliteralen K gehörende Integer-Programm $IP = \Omega(K)$.

2. Falls das 0-1-Integer-Programm IP unlösbar (infeasible) ist,
   dann Ende. K ist unerfüllbar.

3. Wähle eine Lösung L für IP. L ordnet den Atomen in IP Werte aus $\{0, 1\}$ zu, $L : Atome(IP) \rightarrow \{0, 1\}$.

4. Falls es eine Substitution σ und Atome p und q gibt, mit $p\sigma = q\sigma$ und $L(p) \neq L(q)$,
   dann berechne $IP := IP \cup IP\sigma$ und weiter bei 2,
   sonst Ende. K ist erfüllbar.

Falls in Schritt 3 eine Lösung gewählt wird, für die die Bedingung aus Schritt 4 erfüllt ist, dann wird das Integer-Programm um die unter der Substitution σ erzeugten Integer-programming-constraints erweitert. Dazu wird folgendes Beispiel betrachtet:

Beispiel 5.1
Die Klauselmenge K mit

$$K = \{\{p(a)\}, \{p(b)\}, \{q(b)\}, \{\neg p(X), \neg q(X)\}\}$$

wird durch die Transformation Ω in das Integer-Programm IP übersetzt:

$$\begin{aligned} p(a) &\geq 1 \\ p(b) &\geq 1 \\ q(b) &\geq 1 \\ (1 - p(X)) + (1 - q(X)) &\geq 1 \end{aligned}$$

IP besitzt mehrer Lösungen. Es wird die Lösung L gewählt mit $L(p(a)) = L(p(b)) = L(q(b)) = L(q(X)) = 1$ und $L(p(X)) = 0$. Mit der Substitution $\sigma = \{X \leftarrow a\}$ gilt $p(a)\sigma = p(X)\sigma$ und $L(p(a)) \neq L(p(X))$. Nach Anwendung von σ auf IP wird der zusätzliche Integer-programming-constraint

$$(1 - p(a)) + (1 - q(a)) \geq 1$$

zu IP hinzugefügt.

Auch das um diesen neuen Constraint erweiterte Integer-Programm IP ist nicht unlösbar. Unter Verwendung der Lösung $L(p(a)) = L(p(b)) = L(q(b)) = 1$ und $L(q(a)) = L(p(X)) = L(q(X)) = 1$ gilt die Bedingung aus Schritt 4 für $q(X)$ und $q(b)$ mit $\sigma = \{X \leftarrow b\}$. Nun wird

$$(1 - p(b)) + (1 - q(b)) \geq 1$$

hinzugefügt und IP ist damit unlösbar (infeasible). □

Analog zu [BH96] gilt dann auch der folgende Satz, aus dem die Korrektheit und die Vollständigkeit des oben beschriebenen Verfahrens folgen:

Satz 5.2
Die Klauselmenge mit Verzweigungsliteralen K ist unerfüllbar, wenn das 0-1-Integer-Programm $\Omega(K)$ unlösbar (infeasible) ist.

5.3 Bedeutung der zusätzlichen Constraints im Resolutionsverfahren

Die Erzeugung neuer Integer-programming-constraints und deren Hinzufügen zum Integer-Programm (Schritt 4 im oben beschriebene Verfahren) lassen sich nicht direkt mit Schritten in einer Resolutionsableitung vergleichen. Der Ablauf kann auf folgende Weise beschrieben werden:

Es wird versucht, die Resolvente von zwei Klauseln zu bilden. Falls es möglich ist, wird jedoch nicht diese Resolvente erzeugt, sondern nur die dafür notwendige Substitution. Die Substitution wird auf die Klauselmenge angewendet, und die neuen Instanzen der vorhandenen Klauseln werden zur ursprünglichen Klauselmenge hinzugefügt.

Dann kann versucht werden, mit dem herkömmlichen Resolutionsverfahren die leere Klausel abzuleiten. Jedoch darf bei der Bildung einer Resolvente keine Unifikation verwendet werden, um komplementäre Literale zu erzeugen. Es können also nur Resolutionsschritte an Literalen durchgeführt werden, die schon komplementär sind. Läßt sich die leere Klausel nicht herleiten, so werden die beiden Schritte wiederholt.

Es wird also die Unifikation von der eigentlichen Resolventenbildung getrennt, was natürlich nur dann vorteilhaft ist, wenn mit sinnvollen Methoden Nutzen aus den vereinfachten Resolutionsschritten gezogen wird.

6 Erweiterungen und Verfahren

Zunächst werden einige Erweiterungsmöglichkeiten für die vorgestellten Transformationen angegeben (in Abschnitt 6.1). In Abschnitt 6.2 werden das Resolutions- und das Tableauverfahren mit Verzweigungsliteralen bezüglich der Übertragbarkeit von Ableitungsschritten betrachtet. Schließlich werden in Abschnitt 6.3 noch einige abschließende Bemerkungen angegeben.

6.1 Erweiterungen

6.1.1 Ordnen ungeordneter Verzweigungsliterale

Bei der Transformation Φ werden einige, in manchen Fällen auch alle, der ungeordneten Verzweigungsliterale durch die Festlegung der Vergleiche (an den Wahlstellen) geordnet. Diese neuen Vergleiche sind dann für den weiteren Ablauf von Φ bindend. Es ist denkbar, diese Bindung bei der weiteren Berechnung aufzuheben, also bei Vergleichen mit ungeordneten Verzweigungsliteralen jedesmal neu zu entscheiden.

Dadurch können weitere Tableaus dargestellt werden, die mit der vorgestellten Transformation Φ nicht erzeugt werden können. In Bild 6.1 sind die Verzweigungen eines Tableaus T, das dann gebildet werden kann, zu sehen. Dabei gilt für j_2 und j_3 im linken Teilbaum $j_2 >_K j_3$ und im rechten Teilbaum $j_3 >_K j_2$. T kann mit Φ nicht erzeugt werden, da nach dem Aufbau des linken Teilbaums auch im rechten $j_2 >_K j_3$ verwendet wird.

Die Verwendung dieser Darstellung ist jedoch nicht sinnvoll, da die Vorkommen von Verzweigungen, die mit dem gleichen Verzweigungsatom j markiert sind, nicht eine mehrmalige Anwendung einer β-Regel auf die gleiche Ausgangsformel bedeuten. Sie entstehen nur auf Grund der Unbestimmtheit (bezüglich der Ordnung) des Vergleichs von j mit Verzweigungsliteralen, die Verzweigungen oberhalb von j markieren.

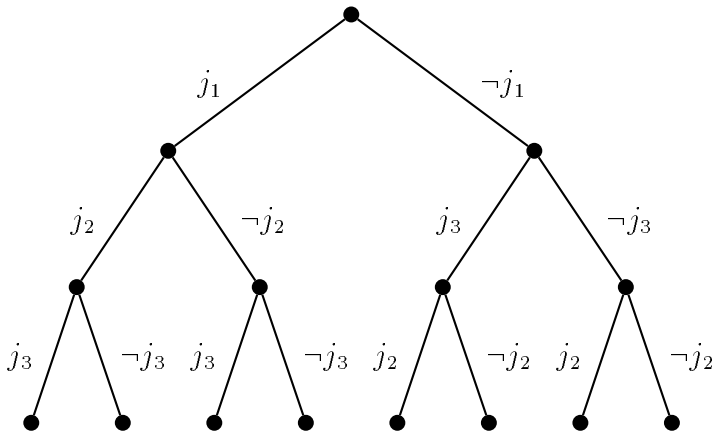


Abbildung 6.1: Das Tableau T.

6.1.2 Weniger Vorkommen von Verzweigungsliteralen

In [BH96] werden bei der Transformation einer prädikatenlogischen Formel in ein Integer-Programm tableaubasierte Regeln verwendet. Eine Formel vom Typ β wird in die Teilformeln β_1 und β_2 zerlegt. Die Markierung von β_1 und β_2 wird gegenüber der von β um eine neue Integer-programming-Variable erweitert (Bild 6.2, linke Seite). Es ist jedoch nicht notwendig, beide Teilformeln von β wieder mit allen Integer-programming-Variablen zu markieren. Die vor β gebildete Summe i kann entweder bei β_1 oder bei β_2 durch 1 ersetzt werden. In Bild 6.2 (rechte Seite) wird bei β_2 die Summe i durch 1 ersetzt.

$$\frac{\boxed{\geq i}\ \beta}{\begin{array}{l}\boxed{\geq i - j(x_1, \ldots, x_n)}\ \beta_1 \\ \boxed{\geq i - (1 - j(x_1, \ldots, x_n))}\ \beta_2\end{array}}$$

Wobei j ein neues Verzweigungsprädikatsymbol ist und $\{x_1, \ldots, x_n\}$ die Schnittmenge der freien Variablen aus β_1 und β_1.

Reduzierte β-Regel:

$$\frac{\boxed{\geq i}\ \beta}{\begin{array}{l}\boxed{\geq i - j(x_1, \ldots, x_n)}\ \beta_1 \\ \boxed{\geq 1 - (1 - j(x_1, \ldots, x_n))}\ \beta_2\end{array}}$$

Wobei j ein neues Verzweigungsprädikatsymbol ist und $\{x_1, \ldots, x_n\}$ die Schnittmenge der freien Variablen aus β_1 und β_1.

Abbildung 6.2: β-Erweiterungsregel aus [BH96] und die reduzierte β-Regel.

Durch die Reduzierung verringert sich zwar die Anzahl der Verzweigungsvariablen nicht, aber die einzelnen Verzweigungsvariablen treten im gebildeten Integer-Programm weniger häufig auf.

Diese Änderung kann analog auch bei der in Abschnitt 2.5.3 beschriebenen Erzeugung von Klauselmengen mit Verzweigungsliteralen verwendet werden. In manchen der gebildeten Klauseln kommen dann weniger Verzweigungsliterale vor. Dabei gehen jedoch die Informationen über den Ast, auf dem β_2 steht, verloren. Werden solche Klauselmengen durch Φ in Tableaus transformiert, dann können Verzweigungen entstehen, bei denen ein Teil leer bleibt. Diese leeren Verzweigungen müssen, nachdem alle Klauseln eingefügt sind, gelöscht werden.

Beispiel 6.1
Eine aus der Formel $F = a \vee (b \wedge (c \vee d))$ erzeugte Klauselmenge K enthält die Klauseln $\{\neg j_1, a\}$, $\{j_1, b\}$, $\{j_1, \neg j_2, c\}$ und $\{j_2, d\}$. Ohne die Verwendung der reduzierten β-Regel wird anstelle von $\{j_2, d\}$ die Klausel $\{j_1, j_2, d\}$ erzeugt. Wenn K durch Φ mit der Ordnung $>_K$ und $j_1 >_K j_2$ in ein Tableau T transformiert wird, dann wird $\{j_2, d\}$ sowohl in den mit j_1 markierten als auch in den mit $\neg j_1$ markierten Teilbaum eingefügt. In beiden Teilbäumen entsteht eine mit j_2 markierte Verzweigung. Das Tableau T ist in Abbildung 6.3 zu sehen. Die unter dem die Formel a enthaltenden Knoten gebildete Verzweigung muß wieder gelöscht werden. □

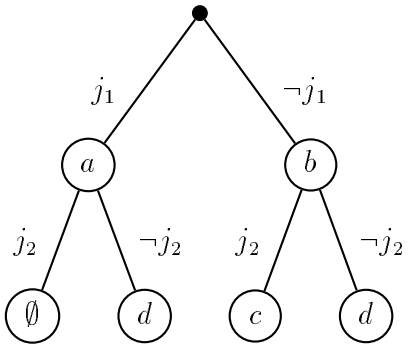


Abbildung 6.3: Das Tableau T.

6.1.3 Tableauabschlüsse

Beim Abschluß eines Asts im Tableau wird die nötige Substitution sofort auf das Tableau angewendet (Definition 2.9). Es ist auch möglich, die Substitution nicht

auszuführen und nur mit dem Abschlußsymbol $\perp$ auf dem Ast zu speichern. Der Ast wird dann noch nicht als geschlossen angesehen. Weitere Regelanwendungen sind möglich. In einem getrennten Schritt kann dann, falls alle Äste mindestens ein Abschlußsymbol enthalten, festgestellt werden, ob es eine Substitution gibt, mit der auf jedem Ast mindestens eine der gespeicherten Substitutionen kompatibel[1] ist. Falls es eine solche gibt, ist das Tableau unter dieser Substitution geschlossen.

Dieses Verfahren kann auch auf Teilbäume angewendet werden. Das Abschlußsymbol mit der Substitution σ kann in einen Multiknoten W eingefügt werden, wenn σ eine gemeinsame Spezialisierung von zwei Substitutionen in den beiden Nachfolgeknoten von W ist.

Es ist auch denkbar, Teile des Tableaus durch Anwendung der Substitutionen zu schließen, wenn feststeht, daß diese Substitutionen auf jeden Fall verwendet werden.

6.2 Vergleich von Verfahren

Die im Anschluß beschriebenen Korrespondenzen zwischen Klauseln mit Verzweigungsliteralen und Formeln im Tableau werden nur für Klauseln aus der starren Teilmenge und den dazugehörenden Tableauformeln mit freien Variablen beschrieben. Die Korrespondenzen gelten aber in gleicher Weise auch für Klauseln aus der universellen Teilmenge. Dann sind die Tableauformeln jedoch universell quantifiziert.

Allgemein gilt, daß die Klausel $\{j_1, \ldots, j_n, l_1, \ldots, l_m\}$ mit den Verzweigungsliteralen $j_1, \ldots, j_n$ und den Formelliteralen $l_1, \ldots, l_m$ im Tableau der Formel $l_1 \vee \ldots \vee l_m$ auf dem Pfad $\overline{j}_1, \ldots, \overline{j}_n$ entspricht. Die mit der Klausel $\{l_1, \ldots, l_m\}$ korrespondierende Formel $l_1 \vee \ldots \vee l_m$ steht im Tableau also an der Wurzel, da keine Verzweigungsliterale vorhanden sind. Falls eine Klausel $\{j_1, \ldots, j_n\}$ keine Formelliterale enthält, bedeutet das im Tableau, daß der mit den Verzweigungsliteralen markierte Ast $\overline{j}_1, \ldots, \overline{j}_n$ geschlossen ist.

6.2.1 Resolventenbildung und Astabschluß

Die Bildung einer Resolvente an den Formelliteralen zweier Klauseln einer Klauselmenge entspricht im Tableau dem Abschluß eines oder mehrerer Äste und

[1] Auf die Berechnung dieser Substitution soll hier nicht näher eingegangen werden. Sie kann durch Spezialisierung aus den auf den Ästen stehenden Substitutionen ermittelt werden. Die Substitution τ ist eine Spezialisierung von σ, wenn es eine Substitution σ' gibt mit $\tau = \sigma' \circ \sigma$.

umgekehrt.

Im einfachsten Fall enthalten die beiden Klauseln C_1 und C_2 jeweils nur ein Formelliteral. Sie können beliebig viele Verzweigungsliterale enthalten. An diesem einen Formelliteral wird unter der Substitution σ die Resolvente D gebildet. In D kommen dann nur Verzweigungsliterale vor. Außerdem sei D keine Tautologie. Im Tableau können alle Äste, die die Verzweigungsliterale aus $\overline{D}$ enthalten, geschlossen werden. Falls D eine Tautologie ist, gibt es im Tableau keine mit D korresponierende Formel. Dieser Fall wird im nächsten Abschnitt 6.2.2 behandelt.

Wenn die Klausel C_1 noch weitere Formelliterale enthält, dann sind diese weiteren Literale auch in der Resolvente D von C_1 und C_2 zu finden. Mit f_1, f_2 und g werden die mit den Klauseln C_1, C_2 und D korrespondierenden Tableauformeln bezeichnet. Bei der Transformation der um die Resolvente D erweiterten Klauselmenge in ein Tableau T wird der Multiknoten, in dem f_1 vorkommt, um die Formel g erweitert. Dabei stehe der Multiknoten von f_1 in T weiter unten als der Multiknoten von f_2, oder f_1 und f_2 kommen im gleichen Multiknoten vor.

Bezüglich des Tableaus ist damit eine Erweiterung vorgenommen worden, die der Ausführung von mehreren Tableauschritten entspricht. Zunächst wird T durch Anwendung einer β-Regel auf f_1 erweitert. Es entstehen zwei neue Blätter B_1 und B_2. In B_1 steht das Literal, das zur Resolventenbildung verwendet wurde. Die zu den restlichen Literalen aus C_1 gehörende Tableauformel steht in B_2. Der Ast zum Blatt B_1 kann mit f_1 und f_2 geschlossen werden. Die dabei benötigte Substitution wird auf T angewendet. Wird der geschlossene Ast aus dem Tableau entfernt, dann erhält man genau das Tableau, das der um die Resolvente D erweiterten Klauselmenge entspricht.

Enthalten beide Klauseln C_1 und C_2 mehrere Formelliterale, so wird das Tableau nacheinander zweimal um eine Verzweigung erweitert, bei der jeweils einer der beiden Äste geschlossen werden kann.

Abbildung 6.4 verdeutlicht den Ablauf für $C_1 = \{p, q\}$, $C_2 = \{\neg q\}$ und $D = \{p\}$.

Eine elegantere Methode besteht darin, komplementäre Formelliterale als Verzweigungsliterale zu betrachten. Die Klauseln $C_1 = \{a, b\}$ und $C_2 = \{\neg a, c\}$ mit der Resolvente $D = \{b, c\}$ von C_1 und C_2 ergeben dann das Tableau T mit

$$T = \langle\{a \vee b\}, ((a, \langle\{c\}, \emptyset\rangle), (\neg a, \langle\{b\}, \emptyset\rangle))\rangle.$$

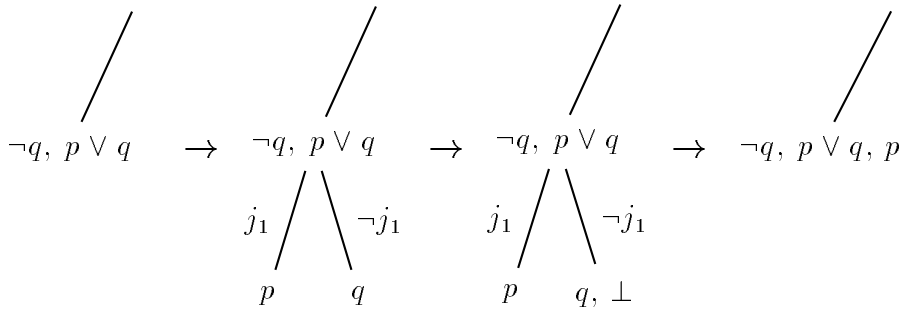


Abbildung 6.4: Die zur Resolventenbildung analogen Schritte im Tableau.

6.2.2 Behandlung von Tautologien

Enthält D zwei komplementäre Literale, dann war es nicht sinnvoll diese Resolvente zu bilden. D ist eine Tautologie, trägt zur Beweisfindung nichts mehr bei und wird nicht benötigt. Auf der Seite der Tableaus bedeutet die Bildung einer Tautologie den Versuch, einen Abschluß mit zwei Literalen auf verschiedenen Tableauzweigen herzuleiten. Dazu wird folgendes Beispiel betrachtet:

Beispiel 6.2
In der Klauselmenge mit Verzweigungsliteralen K mit

$$K = \{\{j_1, j_2, a\}, \{j_1, \neg j_2, b\}, \{\neg j_1, \neg a\}\}$$

wird mit der ersten und dritten Klausel an den Literalen a und $\neg a$ die Resolvente $D = \{j_1, j_2, \neg j_1\}$ gebildet. D ist eine Tautologie. K wird durch Φ (mit $j_1 >_K j_2$) in das Tableau T transformiert mit

$$T = \langle\emptyset, ((\neg j_1, \langle\emptyset, ((\neg j_2, \langle\{a\}, \emptyset\rangle), (j_2, \langle\{b\}, \emptyset\rangle))\rangle), (j_1, \langle\{\neg a\}, \emptyset\rangle))\rangle.$$

T ist auch in Bild 6.5 zu sehen. In T kann kein Abschluß mit a und $\neg a$ herbeigeführt werden, da diese Literale auf verschiedenen Ästen stehen. □

6.2.3 Reguläre Tableaus

Regularität ist eine Einschränkung auf Tableaus, die die Bildung sinnloser identischer Instantiierungen verhindern kann.

Definition 6.3 Reguläres Tableau
Ein Tableau T heißt regulär, wenn auf keinem Ast in T die gleiche Formel mehr als einmal vorkommt.

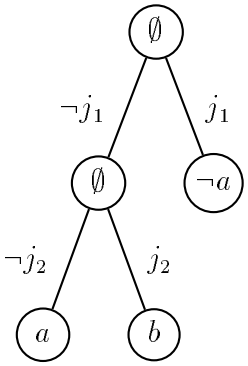


Abbildung 6.5: Das Tableau T, a und $\neg a$ sind auf verschiedenen Ästen.

Durch die Verwendung der Verzweigungsliterale ist es möglich die Regularitätsbedingung auch auf Klauselmengen zu übertragen.

Definition 6.4 Reguläre Klauselmengen mit Verzweigungsliteralen
Eine Klauselmenge mit Verzweigungsliteralen K heißt regulär, falls es keine zwei Klauseln $C_1 \in K$ und $C_2 \in K$ gibt, für die $Vl(C_1) \subseteq Vl(C_2)$ und $Fl(C_1) = Fl(C_2)$ gilt.

6.3 Bemerkungen

Bevor im nächsten Kapitel die Programmierung der Transformationen Φ und Ψ angegeben wird, seien noch folgende abschließende Bemerkungen gemacht. Die in dieser Diplomarbeit eingeführten Transformationen können in vielerlei Hinsicht erweitert werden:

- Es ist denkbar, nicht nur binäre Verzweigungsregeln zu verwenden. Allerdings muß dann die Markierung der Verzweigungen geändert werden. Die Verwendung von einzelnen zweiwertigen Literalen reicht nicht mehr aus. Hier sei besonders auf die Verwendung von Tableaus für mehrwertige Logiken und deren Transformation [Häh94b, Häh92] in Integer-programming-Probleme hingewiesen.

- Bei der Transformation gehen die Informationen darüber, welche logischen Operatoren ($\vee$, $\rightarrow$, ...) den Typ der Formel, an der eine Tableauerweiterung durchgeführt wurde, bestimmt haben, verloren, da auf Resolutionsseite eine konjunktive Normalform benötigt wird. Durch die Speicherung zusammen mit den Verzweigungsliteralen könnten diese Informationen erhalten werden.

- Die Trennung der Klauselmenge in eine universelle und eine starre Teilmenge bietet noch weitere Möglichkeiten. Zum Beispiel bleibt zu klären, in welcher Weise auch Erweiterungen in der universellen Teilmenge vorgenommen werden dürfen.

7 Programmierung

Die Transformationen von Klauselmengen in Tableaus und von Tableaus in Klauselmengen, also Φ und Ψ, wurden in Sicstus-Prolog [AAB+93] programmiert. In Abschnitt 7.1 werden die Hauptprädikate und die Darstellung der Klauseln und Tableaus erläutert. Im darauffolgenden Abschnitt 7.2 ist der Prolog-Programmtext angegeben.

7.1 Realisierung in Prolog

Die Programmierung stimmt im Wesentlichen mit der Beschreibung der Transformationen in den Abschnitten 3.3 und 3.4 überein.

Die Transformation Φ wird durch das Prolog-Prädikat

`built_new_tree_from_clauselist`

realisiert, wobei das erste Tableau, das berechnet werden kann, zurückgegeben wird. Die Unterscheidung der sechs Fälle der Funktion `fe` (Abschnitt 3.3) wird mit `add_to_tree` vorgenommen. Um alle aus einer Klauselmenge und einer Ordnung herstellbaren Tableaus zu erhalten, kann

`all_built_new_tree_from_clauselist`

verwendet werden.

Die Transformation Ψ wird durch

`built_new_clauselist_from_tree`

dargestellt. Aus einem Tableau wird eine Klauselmenge und eine tableaueindeutige Ordnung berechnet. Dabei wird mit `shift_up` die Funktion `lö` aus Abschnitt 3.4 realisiert.

Klauseln werden durch das Prädikat `cl(VzLitLis,LitList)` dargestellt, wobei das erste Argument (`VzLitList`) eine Liste von Verzweigungsliteralen und das

zweite (`LitList`) eine Liste von Formelliteralen ist. Die Tableaus werden wie in Abschnitt 2.4.2 verwendet:

$$\texttt{tree}(\texttt{Formelmenge}, [(\texttt{id}, \texttt{T1}), (-\texttt{id}, \texttt{T2})], \texttt{closed}).$$

`T1` und `T2` sind die mit dem Verzweigungsliteral `id` gekennzeichneten Unterbäume des Multiknotens `Formelmenge`. Mit `closed` kann angegeben werden, ob das Tableau geschlossen ist.

Nun folgt der Programmtext:

7.2 Programmtext

```
% ========================================================================
% Predicates for the set of clauses to tableau transformation
% ========================================================================
% The predicates built_new_tree_from_clauselist, clauses_to_tree,
% add_to_tree are used to built a tree from a given set of
% clauses. add_to_tree is the main predicate.

% ------------------------------------------------------------------------
% built_new_tree_from_clauselist(+ClList,+Ordering,-Tree)
%
% computes one of the possible Trees, that can be built from
% ClList and Ordering.
%
built_new_tree_from_clauselist(ClList,Ordering,Tree):-
        statistics(runtime,[_,_]),
        transitive_closure(Ordering,O2),
        statistics(runtime,[_,Time1]),
        clauses_to_tree(ClList,O2,tree([],[],false),Tree),
        statistics(runtime,[_,Time2]),
        write(Time1),write(' msecs for transitive_closure'),nl,
        write(Time2),write(' msecs for clauses_to_tree'),nl.

% ------------------------------------------------------------------------
% clauses_to_tree(+ClauseList,+Ordering,+Tree,-NewTree)
%
% calls add_to_tree for each clause in ClauseList
%
clauses_to_tree([],_,Tree,Tree).

clauses_to_tree([Fst|Rest],Ordering,Tree,NewTree):-
        add_to_tree(Fst,Ordering,Tree,OutTree,NewOrdering),
```

```
        clauses_to_tree(Rest,NewOrdering,OutTree,NewTree).

% ----------------------------------------------------------------------
% add_to_tree(+Clause,+Ordering,+Tree,-NewTree,-NewOrdering)
%
% extends a tree
% The Lits of the Clause will be added to the Tree according to the
% ordering of the VzLits of the Clause and the NewTree is given back.
% The extended ordering NewOrdering is also given back.
%
add_to_tree(cl([],Lits),Ordering,tree(Fmas,Subtrees,Closed),
                tree(NewFmas,Subtrees,Closed),Ordering):-
        % There are no VzLits: The Lits should be added at the
        % root of the tree
        add_to_fmas(cl([],Lits),Fmas,NewFmas).


add_to_tree(cl(VzLits,Lits),OrderingIn,tree(Fmas,STs,Closed),
                  tree(BackFmas,BackSTList,BackClosed),BackOrdering):-
        % else: go down the tree following the VzLits
        get_max_predicate_from_list(VzLits,OrderingIn,MaxList),
        get_one_maximum_and_add_to_ordering(MaxList,OrderingIn,MaxLit,
                                            Ordering),
        ( STs==[] -> Comp=1 ;   % this will lead to Comp=1 in the case
                     (STs=[(VZ1,ST1),(VZ2,ST2)],   % disjunction below
                     compare(MaxLit,VZ1,Ordering,Comp))),
        (Comp==eq ->
            (remove_vzlit_from_clause(MaxLit,cl(VzLits,Lits),NewCl),
             (VZ1 == MaxLit ->
                (add_to_tree(NewCl,Ordering,ST1,NewST1,BOrd),
                 BackFmas=Fmas,
                 BackSTList=[(VZ1,NewST1),(VZ2,ST2)],
                 BackOrdering=BOrd,
                 BackClosed=Closed);
                (add_to_tree(NewCl,Ordering,ST2,NewST2,BOrd),
                 BackFmas=Fmas,
                 BackSTList=[(VZ1,ST1),(VZ2,NewST2)],
                 BackOrdering=BOrd,
                 BackClosed=Closed)));
         Comp==1 ->
            (remove_vzlit_from_clause(MaxLit,cl(VzLits,Lits),NewCl),
             add_to_tree(NewCl,Ordering,tree([],STs,Closed),NewT,BOrd),
             get_complement(MaxLit,CompMaxLit),
             BackFmas=Fmas,
```

```
            BackSTList=[(MaxLit,NewT),
                        (CompMaxLit,tree([],STs,Closed))],
            BackOrdering=BOrd,
            BackClosed=Closed);
         Comp==2 ->
            (add_to_tree(cl(VzLits,Lits),Ordering,ST1,NewST1,BOrd1),
             add_to_tree(cl(VzLits,Lits),BOrd1,ST2,NewST2,BOrd),
             BackFmas=Fmas,
             BackSTList=[(VZ1,NewST1),(VZ2,NewST2)],
             BackOrdering=BOrd,
             BackClosed=Closed);
         Comp==uc ->
           ((% first alternative: same as Comp=1
             remove_vzlit_from_clause(MaxLit,cl(VzLits,Lits),NewCl),
             get_atfma(MaxLit,MaxLitAt),
             get_atfma(VZ1,VZ1At),
             add_to_ordering(MaxLitAt,VZ1At,Ordering,NewOrdering),
             add_to_tree(NewCl,NewOrdering,tree([],STs,Closed),
                         NewT,BOrd),
             get_complement(MaxLit,CompMaxLit),
             BackFmas=Fmas,
             BackSTList=[(MaxLit,NewT),
                         (CompMaxLit,tree([],STs,Closed))],
             BackOrdering=BOrd,
             BackClosed=Closed);
            (% second alternative: same as Comp=2
             get_atfma(MaxLit,MaxLitAt),
             get_atfma(VZ1,VZ1At),
             add_to_ordering(VZ1At,MaxLitAt,Ordering,NewOrdering),
             add_to_tree(cl(VzLits,Lits),NewOrdering,ST1,NewST1,BOrd1),
             add_to_tree(cl(VzLits,Lits),BOrd1,ST2,NewST2,BOrd),
             BackFmas=Fmas,
             BackSTList=[(VZ1,NewST1),(VZ2,NewST2)],
             BackOrdering=BOrd,
             BackClosed=Closed))).

% ----------------------------------------------------------------------
% all_built_new_tree_from_clauselist(+ClList,+Ordering,-Tree)
%
% the same as built_new_tree_from_clauselist, but this will
% print out all possible trees
%
all_built_new_tree_from_clauselist(ClList,Ordering,Tree):-
        transitive_closure(Ordering,O2),!,
```

```
        clauses_to_tree(ClList,O2,tree([],[],false),Tree),
        write(Tree),nl,nl,
        write_tree(Tree),nl,
        fail.

% ----------------------------------------------------------------------
% add_to_fmas(+Clause,+Fmas,-NewFmas)
%
% Transforms a clause (without branching-literals) in a formula and
% adds it to Fma
%
add_to_fmas(cl(_,Lits),Fmas,NewFmas):-
        clause_to_fma(Lits,F),!,
        append(Fmas,[F],NewFmas).

% clause_to_fma(Lits,F)
clause_to_fma([],mtclause).
clause_to_fma([P],P).
clause_to_fma([Fst|Rest],Lits):-
        clause_to_fma(Rest,Back),
        Lits = Fst v Back.


% ======================================================================
% Predicates for tableau to set of clauses transformation
% ======================================================================

% ----------------------------------------------------------------------
% built_new_clauselist_from_tree(+Tree,-ClauseList,-Ordering)
%
% Transforms a Tree into a ClauseList + Ordering.
%
built_new_clauselist_from_tree(Tree,ClauseList,Ordering):-
        tree_to_clauses([],Tree,ClauseList,[],Ordering).

% ----------------------------------------------------------------------
% tree_to_clauses(+VzLits,+Tree,-ClauseList,+InOrd,-OutOrd)
%
% Builts the clauses from the formulas of Tree. The branching-
% literals are collected in VzLits, while going down the tree.
% They are added to the clauses.
% OutOrd is the ordering computed from Tree.
%
tree_to_clauses(VzLits,tree(Fmas,[],_),ClauseList,Ord,Ord):-
```

```
        fmas_to_clauses(Fmas,VzLits,ClauseList).
tree_to_clauses(VzLits,tree(Fmas,[(VZ1,ST1),(VZ2,ST2)],_),ClauseList,
                InOrd,OutOrd):-
        fmas_to_clauses(Fmas,VzLits,Back3),
        tree_to_clauses([VZ1|VzLits],ST1,Back1,InOrd,Ord1),
        tree_to_clauses([VZ2|VzLits],ST2,Back2,Ord1,Ord2),
        get_atfma(VZ1,VZAt),
        get_atfma_list(VzLits,VzListAt),
        add_to_ordering_list1(VzListAt,VZAt,Ord2,Ord3),
        transitive_closure(Ord3,OutOrd),
        shift_up(VZ1,VZ2,Back1,Back2,NewBack1,NewBack2,OutOrd),
        append(NewBack1,NewBack2,B),
        append(B,Back3,ClauseList).

% ------------------------------------------------------------------------
% shift_up(+VZ1,+VZ2,+ClList1,+ClList2,-NewClList1,-NewClList2,+Ord)
%
% If there are two equal formulas (clauses) (F,G) at branches whose
% branching-literals are only different at VZ1 and VZ2, one of these
% formulas can be deleted and the branching-lieral (VZ1 or VZ2) can
% be deleted in the other formula.
%
shift_up(_,_,[],ClList2,[],ClList2,_):-!.
shift_up(VZ1,VZ2,[cl(VzLits,Lits)|Rest1],ClList2,[NFst1|NewRest1],
         NewClList2,Ord):-
        delete(VzLits,VZ1,NVzLits),
        delete(NVzLits,VZ2,RedVzLits),
        shift_up_loop(VZ1,VZ2,cl(VzLits,Lits),cl(RedVzLits,Lits),
                      ClList2,NFst1,NClList2,Ord),
        shift_up(VZ1,VZ2,Rest1,NClList2,NewRest1,NewClList2,Ord),!.

% shift_up_loop(+VZ1,+VZ2,OrgCl1,RedCl1,CLList2,NewCl1,NewClList2)
%
% If we can find a clause in ClList2 whose reduced form (without VZ1
% or VZ2) is equal to RedCl1, then we will remove this clause from
% ClList2 an take RedCl1 instead of OrgCl1.
%
shift_up_loop(_,_,Cl1,_,[],Cl1,[],_):-!.
shift_up_loop(VZ1,VZ2,cl(VzLitsOrgCl1,LitsOrgCl1),RedCl1,
              [cl(VzLits,Lits)|Rest2],NewCl1,NewClList2,Ord):-
        delete(VzLits,VZ1,NVzLits),
        delete(NVzLits,VZ2,RedVzLits),
        get_min_predicate_from_list(VzLitsOrgCl1,Ord,Min1),
        get_min_predicate_from_list(VzLits,Ord,Min2),
```

```
        get_atfma_list(Min1,Min1Atfma),
        get_atfma_list(Min2,Min2Atfma),
        ord_union(Min1Atfma,Min2Atfma,Mins),
        get_atfma(VZ1,VZ1At),
        ( ((RedCl1 == cl(RedVzLits,Lits)),
           nu_memberchk_not_in(VZ1At,Mins)) ->
                (NewCl1=RedCl1,
                 NewClList2=Rest2);
            % else
                (shift_up_loop(VZ1,VZ2,cl(VzLitsOrgCl1,LitsOrgCl1),
                               RedCl1,Rest2,NCl1,NRest2,Ord),
                 NewCl1=NCl1,
                 NewClList2=[cl(VzLits,Lits)|NRest2])),!.

% ----------------------------------------------------------------------
% fmas_to_clauses(+Fmas,+VzLits,-ClauseList)
%
% Transforms a list of formulas in alist of clauses.
%
fmas_to_clauses([],_,[]).
fmas_to_clauses([Fst|Rest],VzLits,[cl(VzLits,Cl)|RestCl]):-
        fma_to_clause(Fst,Cl),
        fmas_to_clauses(Rest,VzLits,RestCl).

% fma_to_clause(+Fma,-Clause)
%
% Transforms the Fma a v b v c in Clause [a,b,c]
%
fma_to_clause(F v G,[F|List]):-
        fma_to_clause(G,List),!.
fma_to_clause(F,[F]).


% ======================================================================
% Predicates to manage the ordering
% ======================================================================

% ----------------------------------------------------------------------
% transitive_closure(+Ord,-TC)
%
% Computes the transitive closure TC of Ord.
%
transitive_closure(OrdList,NewOrdList):-
        list_to_ord_set(OrdList,OrdList1),
```

```
        expand_loop_1(OrdList1,[],NewOrdList),!.

% expand_loop_1(+OrdList,+NewOrdList,-NewOrdListNew)
expand_loop_1([],L,L).
expand_loop_1([Fst|Rest],[],NewOrd):-
        expand_loop_1(Rest,[Fst],NewOrd).
expand_loop_1([Fst|Rest],List,NewOrd):-
        expand_loop_2(Fst,List,AddtoRest),
        ord_union(Rest,AddtoRest,NewRest),
        ord_add_element(List,Fst,List2),
        expand_loop_1(NewRest,List2,NewOrd).

% expand_loop_2(+OComp,+NewOrdList,-NewOComps)
expand_loop_2(_,[],[]).
expand_loop_2(o(A,B),[o(B,C)|Rest],AddtoOrd2):-
        expand_loop_2(o(A,B),Rest,AddtoOrd),
        ord_add_element(AddtoOrd,o(A,C),AddtoOrd2).
expand_loop_2(o(A,B),[o(C,A)|Rest],AddtoOrd2):-
        expand_loop_2(o(A,B),Rest,AddtoOrd),
        ord_add_element(AddtoOrd,o(C,B),AddtoOrd2).
expand_loop_2(X,[_|Rest],AddtoOrd):-
        expand_loop_2(X,Rest,AddtoOrd).

% ------------------------------------------------------------------------
% compare(+Elem1,+Elem2,+Ordering,-Out)
%
% Compares the predicate-symbols of Elem1 and Elem2 according to
% the Ordering. Gives back
% eq if Elem1=Elem2
% 1  if Elem1>Elem2
% 2  if Elem1<Elem2
% uc if Elem1 and Elem2 are uncompareable
% (For every ordering O and every e in O: e>ordbottom)
%
compare(E1,E2,O,Out):-
        ccompare(E1,E2,O,Out),!.

ccompare(ordbottom,_,_,2).
ccompare(_,ordbottom,_,1).

ccompare(-Elem1,Elem2,Ordering,Out):-
        ccompare(Elem1,Elem2,Ordering,Out).
ccompare(Elem1,-Elem2,Ordering,Out):-
        ccompare(Elem1,Elem2,Ordering,Out).
```

```
ccompare(Elem1,Elem1,_,eq).
ccompare(Elem1,Elem2,Ordering,1):-
        functor(Elem1,F1,_),
        functor(Elem2,F2,_),
        nu_memberchk(o(F1,F2),Ordering).
ccompare(Elem1,Elem2,Ordering,2):-
        functor(Elem1,F1,_),
        functor(Elem2,F2,_),
        nu_memberchk(o(F2,F1),Ordering).
ccompare(_,_,_,uc).

% ------------------------------------------------------------------------
% get_max_predicate_from_list(+List,+Ordering,-MaxList)
% get_min_predicate_from_list(+List,+Ordering,-MinList)
%
% Get the maximal predicates (according to Ordering) from
% List. If the ordering is total then MaxList=[t], where t is
% the maximum.
% Same for minimum.
%
get_max_predicate_from_list([],_,[]).
      % this case should not appear
get_max_predicate_from_list([Fst|Rest],Ordering,Max):-
        search_for_max(Fst,Rest,Ordering,Max).

% search_for_max(+P,+List,+Ordering,-Max)
search_for_max(P,[],_,[P]).
search_for_max(P,[Fst|Rest],Ordering,Max):-
        compare(P,Fst,Ordering,Out),
        ( Out = 1  -> search_for_max(P,Rest,Ordering,Max);
          Out = 2  -> search_for_max(Fst,Rest,Ordering,Max);
          Out = uc -> (search_for_max(P,Rest,Ordering,Max1),
                       search_for_max(Fst,Rest,Ordering,Max2),
                       ord_union(Max1,Max2,Max))).

get_min_predicate_from_list([],_,[]).
      % this case should not appear
get_min_predicate_from_list([Fst|Rest],Ordering,Min):-
        search_for_min(Fst,Rest,Ordering,Min).

% search_for_min(+P,+List,+Ordering,-Min)
search_for_min(P,[],_,[P]).
search_for_min(P,[Fst|Rest],Ordering,Min):-
```

```
        compare(P,Fst,Ordering,Out),
        ( Out = 2  -> search_for_min(P,Rest,Ordering,Min);
          Out = 1  -> search_for_min(Fst,Rest,Ordering,Min);
          Out = uc -> (search_for_min(P,Rest,Ordering,Min1),
                       search_for_min(Fst,Rest,Ordering,Min2),
                       ord_union(Min1,Min2,Min))).

% ------------------------------------------------------------------
% nu_memberchk(Elem,OrdSet)
%
% Checks if Elem is in OrdSet. nu_memberchk does no unification (nu):
% nu_memberchk(p(X),[p(a)]) and nu_memberchk(X,[a,b,c]) are failing.
%
nu_memberchk(Elem,OrdSet):-
        (is_ordset(OrdSet) ->
             ord_subset([Elem],OrdSet);
            (% should not appear
             write('Warning: Set in nu_memberchk should be an OrdSet'),
             nl,
             write('         (nu_memberchk fails)'),nl,
             fail)).

nu_memberchk_not_in(Elem,OrdSet):-
        (nu_memberchk(Elem,OrdSet) -> fail;true).

% ------------------------------------------------------------------
% add_to_ordering(+Vz1,+Vz2,+InOrd,-OutOrd)
% add_to_ordering_list1(+Vz1List,+Vz2,+InOrd,-OutOrd)
% add_to_ordering_list2(+Vz1,+Vz2List,+InOrd,-OutOrd)
%
% adds o(Vz1,Vz2) to the ordering InOrd
% If Vz1 is a list, then o(E,Vz2) will be added to InOrd, for all
% Elements E of the List Vz1.
% If Vz2 is a list, then o(Vz1,E) for all E in Vz2.
%
add_to_ordering(Vz1,Vz2,InOrd,OutOrd):-
        ord_add_element(InOrd,o(Vz1,Vz2),OutOrd).

add_to_ordering_list1([],_,InOrd,InOrd).
add_to_ordering_list1([Vz1|Rest],Vz2,InOrd,OutOrd):-
        ord_add_element(InOrd,o(Vz1,Vz2),OutOrd1),
        add_to_ordering_list1(Rest,Vz2,OutOrd1,OutOrd).

add_to_ordering_list2(_,[],InOrd,InOrd).
```

```
add_to_ordering_list2(Vz1,[Vz2|Rest],InOrd,OutOrd):-
        ord_add_element(InOrd,o(Vz1,Vz2),OutOrd1),
        add_to_ordering_list2(Vz1,Rest,OutOrd1,OutOrd).


% ======================================================================
% Further Predicates
% ======================================================================

:- use_module(library(lists)).
:- use_module(library(ordsets)).
:- op( 500,fx,-).
:- op( 550,xfy,&).
:- op( 560,xfy,v).
:- op( 570,xfy,=>).
:- op( 580,xfy,<=>).

% ----------------------------------------------------------------------
% get_complement(Lit,OutLit)
%
% Gives back the complement of literal Lit
%
get_complement(-Lit,Lit):-!.
get_complement(Lit,-Lit):-!.

% ----------------------------------------------------------------------
% get_atfma(+Lit,-AtFma)
% get_atfma_list(+Lit,-AtFma)
%
% p(X) -> p(X),  -p(X) -> p(X)
%
get_atfma(-X,X):-!.
get_atfma(X,X):-!.

get_atfma_list([],[]).
get_atfma_list([F|R],Out):-
        get_atfma(F,F1),
        get_atfma_list(R,R1),
        ord_add_element(R1,F1,Out).

% ----------------------------------------------------------------------
% remove_vzlit_from_clause(+Lit,+Cl,-NewCl)
%
% removes the branching-literal Lit from the clause Cl.
```

```
%
remove_vzlit_from_clause(Lit,cl(VzLits,Lits),cl(NewVzLits,Lits)):-
        delete(VzLits,Lit,NewVzLits).

% ----------------------------------------------------------------------
% get_one_maximum(+List,-Max)
%
% Choose one of the maxima in List. If the ordering is total there
% will be no choice, because List contains only one maximum.
% Otherwise we have a CHOICEPOINT.
% The first element of List is taken at the moment.
%
get_one_maximum([P|_],P).


% ----------------------------------------------------------------------
% get_one_maximum_and_add_to_ordering(+List,+Ord,-Max,-NewOrd)
%
% Choose one of the maxima in List. If the ordering is total there
% will be no choice, because List contains only one maximum.
% Otherwise we have a CHOICEPOINT.
% The first element of List is taken at the moment.
% Fixes the ordering.
%
get_one_maximum_and_add_to_ordering([P|Rest],InOrd,P,OutOrd):-
        add_to_ordering_list2(P,Rest,InOrd,OutOrd).


% ----------------------------------------------------------------------
% get_predicate_symbol_from_literal(+Literal,-PSymbol)
%
% Gives back the predicatesymbol of Literal. Example: -p(x) -> p
%
get_predicate_symbol_from_literal(-Literal,PSymbol):-
        get_predicate_symbol_from_literal(Literal,PSymbol).
get_predicate_symbol_from_literal(Literal,PSymbol):-
        functor(Literal,PSymbol,_).


% ======================================================================
% Output predicates
% ======================================================================

write_cl_list([]):-
```

```
        nl.
write_cl_list([Fst|Rest]):-
        write_cl(Fst),
        nl,
        write_cl_list(Rest),!.

write_cl(cl(L1,L2)):-
        write('{'),
        write_cl_loop(L1),
        (L1 == [] -> true; write(',')),
        write_cl_loop(L2),
        write('}').

write_cl_loop([]).
write_cl_loop([Fst]):-
        write(Fst).
write_cl_loop([Fst|Rest]):-
        write(Fst),write(','),
        write_cl_loop(Rest).

% ---------------------------------------------------------------------
write_fmas_loop([]):-
        write('*'),nl.
write_fmas_loop([Fst]):-
        write(Fst),nl.
write_fmas_loop([Fst|Rest]):-
        write(Fst),write(' , '),
        write_fmas_loop(Rest).

% ---------------------------------------------------------------------
write_tree(Tree):-
        write_tree_loop(Tree,''),!.

write_tree_loop(tree(Fmas,[],_),_):-
        write_fmas_loop(Fmas).

write_tree_loop(tree(Fmas,[(J1,ST1),(J2,ST2)],_), Shift):-
        name(Shift,HelpShift),
        append("    ",HelpShift,HelpShift2),
        name(Shift2,HelpShift2),
        write_fmas_loop(Fmas),!,
        write(Shift),write(J1),write(': '),
        write_tree_loop(ST1,Shift2),!,
        write(Shift),write(J2),write(': '),
```

```
        write_tree_loop(ST2,Shift2).


% ---------------------------------------------------------------------
% tree_to_psbeta(Tree,FileName)
%
tree_to_psbeta(Tree,FileName):-
        open(FileName,write,Stream),
        tree_to_psbeta_loop(Stream,Tree),
        nl(Stream),
        close(Stream).

% tree_to_psbeta_loop(Stream,Tree)
tree_to_psbeta_loop(Stream,tree(W,[],_)):-
        psnode(Stream,W).
tree_to_psbeta_loop(Stream,tree(W,[(_,ST1),(_,ST2)],_)):-
        write(Stream,'\pstree{'),
        psnode(Stream,W),
        write(Stream,'}{%'),
        nl(Stream),
        tree_to_psbeta_loop(Stream,ST1),
        nl(Stream),
        tree_to_psbeta_loop(Stream,ST2),
        write(Stream,'}').
psnode(Stream,W):-
        write(Stream,'\Tr{'),
        psnode_loop(Stream,W),
        write(Stream,'}').

psnode_loop(_,[]).
psnode_loop(Stream,[A]):-
        write(Stream,A).
psnode_loop(Stream,[F|R]):-
        write(Stream,F),
        write(Stream,','),
        psnode_loop(Stream,R).
```

Abbildungsverzeichnis

Literaturverzeichnis

[AAB+93] Johan Anderson, Stefan Anderson, Kent Boortz, Hans Nilsson und Thomas Sjöland. *SICStus Prolog 2.1 User's Manual.* Swedish Institute of Computer Science, Januar 1993.

[Bet86] E. W. Beth. Semantic Entailment and Formal Derivability. In Karel Berka und Lothar Kreiser, Hrsg., *Logik-Texte. Kommentierte Auswahl zur Geschichte der modernen Logik*, Seiten 262–266. Akademie-Verlag, Berlin, 1986.

[BH96] Bernhard Beckert und Reiner Hähnle. Deduction by Combining Semantic Tableaux and Integer Programming. In *Proceedings, Annual Conference of the European Association for Computer Science Logic (CSL'95), Paderborn, Germany*, LNCS. Springer, 1996.

[CL73] Chin-Liang Chang und Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, London, 1973.

[Fit90] Melvin C. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer, New York, 1990.

[Häh92] Reiner Hähnle. A New Translation from Deduction into Integer Programming. In Jacques Calmet und John A. Campbell, Hrsg., *Proc. Int. Conf. on Artificial Intelligence and Symbolic Mathematical Computing AISMC-1, Karlsruhe, Germany*, Seiten 262–275. Springer, LNCS 737, 1992.

[Häh94a] Reiner Hähnle. Automatisches Beweisen. Vorlesungsskriptum. Fakultät für Informatik, Universität Karlsruhe, Sommersemester 1994.

[Häh94b] Reiner Hähnle. Many-Valued Logic and Mixed Integer Programming. *Annals of Mathematics and Artificial Intelligence*, 12(3,4):231–264, Dezember 1994.

[Hoo92a] John N. Hooker. Generalized Resolution for 0–1 Linear Inequalities. *Annals of Mathematics and Artificial Intelligence*, 6:271–286, 1992.

[Hoo92b] John N. Hooker. New Methods for Computing Inferences in First Order Logic. Working Paper, GSIA, CMU Pittsburgh, April 1992.

[Hoo93] John N. Hooker. Logic-Based Methods for Optimization. A Tutorial. Working Paper, GSIA, CMU Pittsburgh, Dezember 1993.

[Lei92] Alexander Leitsch. Resolution Theorem Proving. Lecture Notes of Fourth Summer School on Language, Logic and Information, Colchester/England, August 1992.

[Ree87] Steve Reeves. Semantic Tableaux as a Framework for Automated Theorem-Proving. In C. S. Mellish und J. Hallam, Hrsg., *Advances in Artificial Intelligence (Proceedings of AISB-87)*, Seiten 125–139. Wiley, 1987.

[Rob65] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *JACM*, 12(1):23–41, Januar 1965. Nachdruck in [SW83].

[Sch87] Peter H. Schmitt. The THOT Theorem Prover. Technical Report TR–87.09.007, IBM Heidelberg Scientific Center, 1987.

[Sch89] Uwe Schöning. *Logik für Informatiker.* BI-Wissenschafts Verlag, Mannheim, 2., überarb. Auflage, 1989.

[SM89] Harvey Salkin und Kamlesh Mathur. *Foundations of Integer Programming.* North-Holland, 1989.

[Smu68] Raymond M. Smullyan. *First-Order Logic.* Springer, Berlin, 1968.

[SW83] Jörg Siekmann und Graham Wrightson, Hrsg. *Automation of Reasoning: Classical Papers in Computational Logic 1957–1966*, Jgg. 1. Springer-Verlag, 1983.