Konnektions-Kalküle

Einführung in Konnektions-Tableaus

Seminar Inferenzmethoden – SS 2005
Martin Hilscher

1. Kurzzusammenfassung

Diese Ausarbeitung hat die Aufgabe die Grundlagen für das Verständnis der Konnektions-Tableaus zu liefern. Dabei werden zu Beginn grundlegende Begriffe sowie deren Verwendung erläutet, welche ihre Gültigkeit auch über diesen Abschnitt *„Einführung in Konnektions-Tableaus“* hinweg aufrechterhält. Anschließend wird der Aufbau eines Konnektions-Tableaus betrachtet, dann wie man in diesem die Beweissuche durchführt und abschließend wird sich mit der Frage beschäftigt wie man sich bei der Teilzielwahl erhält.

2. Vorbetrachtung / Begriffserklärung

Bevor sich dem Tableau Beweis zugewandt werden kann, ist es sinnvoll einige grundlegende Begriffe und Notationen nochmals kurz zu erläutern. Dies hat den Vorteil, dass der spätere Ablauf nicht durch eingeworfene Beschreibungen oder Definitionen gestört wird.
Im Folgenden werden freie Variablen immer mit den kleinen Buchstaben u bis z belegt, während gebundene Variablen bzw. Konstanten die kleinen Buchstaben von a bis d tragen. Als Funktionssymbole dienen die kleinen Buchstaben f, g, und h, Prädikatensymbole tragen die Grossbuchstaben P, Q oder R. Des Weiteren werden die üblichen Symbole für die logischen Verknüpfungen UND und ODER sowie die Negation verwendet. Weiterhin finden der All- sowie der Existenzquantor Verwendung. Zusammenfassend werden also folgende Symbole verwendet:

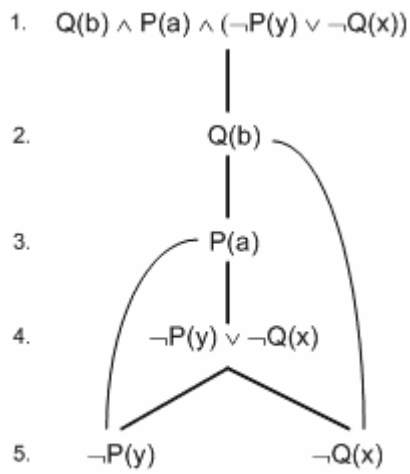
Variablen:	u, v, w, x, y, z
Konstanten:	a, b, c, d
Funktionen:	f, g, h
Prädikatensymbole:	P, Q, R
Sonstige:	$\wedge, \vee, \neg, \forall, \exists$, (,)

Da nun die grundlegenden Symbole bekannt sind, sollen an dieser Stelle kurz die wichtigsten Grundbegriffe im Umgang mit den Tableau Beweisen besprochen werden. Den Grundstein bildet der Term, welcher entweder aus einer Variable oder einem n-stelliges Funktionssymbol α der Form $\alpha(t_1,\ldots,t_n)$ besteht. Wobei t_i wiederum Terme sind. Eine atomare Formel ist ein Gebilde, welches nicht weiter zerlegbar ist und hat die Form: $\alpha(t_1,\ldots,t_n)$ wobei t_i Terme sind und α ein Prädikatensymbol. Als Literal wird eine atomare Formel bezeichnet die möglicherweise noch mit einer Negation versehen ist. Eine Klausel ist eine Disjunktion von Literalen, handelt es sich um nur ein Literal so wird diese Klausel auch Unit Clause genannt. Ein weiterer Spezialfall sind die so genannten Hornklauseln, welche höchstens ein positives Literal enthalten. Im Allgemeinen haben Klauseln die Form: $(L_1 \vee L_2 \vee L_3 \vee \ldots \vee L_n)$. Das größte Konstrukt bildet letztendlich die Formel, welche aus einer Konjunktion von Klauseln, mit der Form $(K_1 \wedge K_2 \wedge K_3 \wedge \ldots \wedge K_{n)}$ besteht. Da es sich bei Konnektions-Tableaus ausschließlich um Klausel-Tableaus handelt, haben demzufolge alle Formeln die oben genannte Form.
Ein kleine Zusammenfassung der eben genannten Begriffe erfolgt auf der kommenden Seite:

Term:	x oder f(x,y)
Atomare Formel:	P(a,b)
Literal:	¬P(a)
Klausel:	(a ∨ x ∨ ¬y) oder ¬x als Unit Clause
Formel:	(a ∨ x ∨ ¬y) ∧ (¬P(a))

3. Aufbau des Konnektions-Tableaus

Bei einem Klausel-Tableau Beweis handelt es sich immer um eine indirekte Beweisführung, d.h. man versucht zu zeigen, dass die Ausgangsformel F, negiert zu ¬F, immer zu einem Widerspruch führt. Der eigentliche Beweis wird dabei an einem Baum geführt, beginnend bei der (negierten) Ausgangsformel. Das negieren der Ausgangsformel hat den einfachen Grund, dass wenn man mit ¬F beginnt so den erhofften Widerspruch aufzeigen kann. Die einzelnen Klauseln der Formel, welche durch Konjunktionen miteinander verbunden sind, werden nun untereinander in einer Baumstruktur an die Ausgangsformel angehangen. Diesen Schritt stellen die ersten vier Zeilen in der unten folgenden Abbildung 1 dar, welche mit einem Beispiel den Sachverhalt veranschaulichen soll. Um den Konnektions-Tableau Beweis nun fortzuführen bzw. beginnen zu können, werden gewisse Beweisregeln angewendet.



σ = [a|y , b|x]

Abbildung 1

Zum einen gibt es die Expansionsregel, welche den Beweis verzweigt. Dies erfolgt indem man eine Klausel an seinen Disjunktionen in unterschiedlichen Ästen aufspaltet. In der Abbildung 1 wurde in der 4. Zeile die Klausel $\neg P(y) \vee \neg Q(x)$ an der Disjunktion zu zwei neuen Ästen $\neg P(y)$ und $\neg Q(x)$ aufgespaltet. Die neuen Äste werden dabei immer an den Ast angehangen an dem sich die ursprüngliche Klausel befand.

Die Reduktionsregel liefert uns dagegen ein Teilziel. Diese Regel besagt, dass wenn ein Ast so instanziiert werden kann, dass das Komplement zu einem seiner Vorgängerknoten gefunden ist, so ist dieser Ast abgeschlossen. Diese Verbindung zwischen einem Ast und seinem Komplement in einem Vorgängerknoten werden mittels Konnektionen hergestellt. Konnektionen sind im Wesentlichen nur Linien die den Bezug vom Ast zu seinem Komplement visualisieren sollen. Jedoch unterscheidet man in zwei Arten von Konnektionen. Zum einen in Pfad-Konnektionen, welche die Konnektion eines Astes mit irgendeinem Vorfahren beschreibt. Diese Art wird auch als schwache Konnektion bezeichnet. Während die andere Konnektion, welche zu dem direkten Vorgänger geschlagen wird, als starke Konnektion oder einfach nur als Konnektion bezeichnet. Die Anwendung der Reduktionsregel und die damit verbundenen Konnektionen werden ebenfalls in Abbildung 1 gezeigt. Hierbei ist zu beachten, dass y mit a und x mit b unifiziert werden, um die benötigten Komplemente zu bilden.
Ein Beweis zählt als abgeschlossen, wenn alle seine Äste wie oben beschrieben abgeschlossen wurden. Es wurde dann nämlich gezeigt, dass ¬F immer zum Widerspruch führt und damit muss zwängläufig F gelten.

Das Ziel des Konnektions-Tableaus ist natürlich die Automatisierung, um auch Probleme zu lösen die für den Menschen zu groß sind oder aber zu lange dauern würden. Deswegen wurden die Regeln in der Extensionsregel zusammengefasst, in der jeder Beweisschritt „im Zusammenhang“ mit dem vorhergehenden stehen soll. Diese Regel besagt, dass jeder Expansionsschritt direkt von einem Reduktionsschritt gefolgt sein muss, um so ein zielorientiertes, maschinelles Arbeiten durch den Beweis zu gewährleisten. D.h. also, dass es zu jedem Knoten, der einen Nachfolger und die Markierung *L* hat, einen Nachfolger gibt, dessen Markierung zu *L* komplementär ist. Dabei eröffnet sich aber leider ein nicht zu unterschätzendes Problem. Konnektions-Tableaus sind nicht beweiskonfluent, bei unzweckmäßiger Wahl der Klauseln, mit denen das Tableau erweitert wird, kann unter Umständen kein Beweis gefunden werden. Bei fehlender Beweiskonfluenz, wie es bei den Konnektions-Tableaus der Fall ist, müssen bei der Beweissuche systematisch alle möglichen Tableaus aufgezählt werden. Betrachtet man die Klauselmenge {¬P(a), R(x) ∨ Q(x), ¬R(b), P(x) ∨ Q(x), ¬Q(a)} so kann diese ganz leicht durch die Klauselerzeugung von P(x) ∨ Q(x) begonnen werden und dann gemäß Extensionsregel jeder der zwei sich ergebenden Äste P(x) und Q(x) durch ihr jeweiliges Komplement ¬P(a) bzw. ¬Q(a) abgeschlossen werden. Würde man jedoch mit R(x) ∨ Q(x) beginnen, so könnte man den Beweis nicht abschließen, obwohl es wie oben gezeigt eine Lösung des Problems gibt. Demzufolge muss das System alle möglichen Deduktionen durchlaufen. Das sprengt jedoch zwangsläufig die Größe des Speichers, wenn man bedenkt, dass sich bei jeder Expansion immer mehr Möglichkeiten ergeben. Es müssen sinnvolle Methoden und Beschränkungen gefunden werden um die Suche einzuschränken. Mit dieser Problematik wird sich der nächste Abschnitt beschäftigen.

4. Beweissuche im Konnektions-Tableau

Um der automatischen Beweissuche gewisse Schranken aufzuerlegen und damit ein Überlaufen den Speichers zu vermeiden, werden so genannte Grenzen eingeführt, welche das automatisierte System nicht überschreiten darf.
Eine dieser Beschränkungen ist die Inferenz-Grenze. Diese gibt an, wie viele Inferenzschritte das System maximal machen darf. Die Abarbeitung erfolgt dabei stufenweise, d.h. alle offenen Teilziele (einer Ebene) werden einen Schritt vorangebracht und nicht erst ein Teilziel bis zum Ende verfolgt. In die Berechnung werden auch die neu angehangenen Klauseln im folgenden Sinne miteinbezogen. Die Länge der jeweiligen Klausel wird dazugezählt, da man davon ausgehen muss, dass für jedes Literal mindestens ein Inferenzschritt zum lösen benötigt wird. Für ein Literal der Länge drei, beispielsweise *(x ∨ y ∨ z)*, ergeben sich nach dem Expansionsschritt drei neue Teilziele bzw. Äste. Um diese zu lösen werden wenigstens drei Schritte benötigt. Die Abbildung 2 soll die Idee hinter dieser Grenze verdeutlichen.

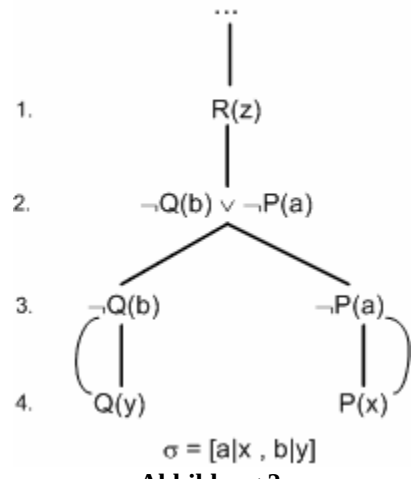


Abbildung 2

In der ersten Zeile sind noch n Schritte erlaubt. In der zweiten eigentlich noch n-1, doch da die angehangene Klausel die Länge 2 hat, sind es nur noch n-3 Schritte. In der 3. Zeile wurde das Literal nun an der Disjunktion gemäß der Expansionsregel aufgespaltet. Beide Äste konnten in der 4. Zeile abgeschlossen werden und so sind insgesamt noch n-3 Schritte verblieben. Die Zahl ändert sich nicht mehr, da bereits die minimalen Kosten zum lösen der beiden Teilziele miteinbezogen wurden. Wäre jetzt beispielsweise ein Ast noch nicht abgeschlossen und man könnte nur eine Klausel anhängen deren Länge

größer als die verbleibenden Schritte sind, so würde das System abbrechen. Zur Optimierung dieses Verfahrens kann man die „Divide & Conquer“ Methode zur Erforschung der Teilziele einsetzen, was aber natürlich erhöhten Implementationsaufwand bedeutet.

Ein anderer Ansatz besteht darin, die Grenze nicht an Hand der Inferenzschritte auszumachen, sondern die maximale Tiefe des Baums zu berücksichtigen. Diese Beschränkung wird Tiefengrenze genannt. Diese Grenze ist jedoch recht ungenau, da sich der Baum in der Regel je Tiefer dieser wird immer mehr verzweigt.

Die nebenstehende Abbildung 3 zeigt die Problematik der Verzweigungen mit zunehmender Tiefe und verdeutlicht gleichzeitig die Zählweise. Eine mögliche Optimierungsvariante ist die Tiefengrenze unter Berücksichtigung der jeweiligen Klauseln. Hier werden die Werte der einzelnen Knoten über eine Funktion berechnet, anstatt diese nur abhängig von der Tiefe des Baumes zu machen. In diese Funktion fließt sowohl die aktuelle Tiefe sowie die Anzahl der neu hinzu gekommenen Teilziele durch anhängen einer Klausel ein. Die genaue Implementation ist jedoch vom System abhängig.

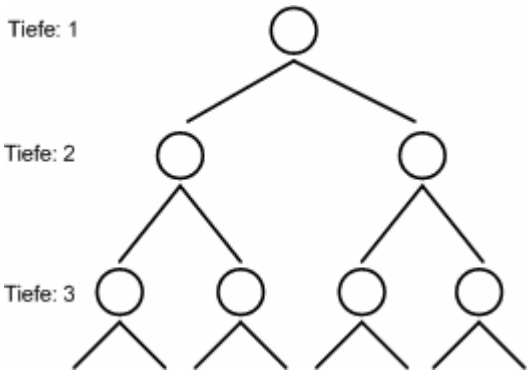


Abbildung 3

Eine andere Möglichkeit zur Optimierung bietet die gewichtete Tiefengrenze. Diese ist eine Kombination aus Inferenz- und Tiefengrenze. Als Basis gilt die Tiefengrenze und die Inferenzgrenze wird mit einbezogen. Der Wert eines Knoten wird dabei genau wie bei der klauselabhängigen Tiefengrenze mittels einer Funktion berechnet, in der höchstens die Gewichtung etwas anders verteilt ist. Dieser Wert wird dann in einem garantierten Anteil und einen zusätzlichen Anteil unterteilt. Der garantierte Anteil ist der Teil der zur Lösung des Teilsziels auf jeden Fall verbraucht werden kann. Der zusätzliche Teil ist eine Art gemeinsame Ressource und immer wenn ein Teilziel gelöst wurde, wird der zusätzliche Teil bei allen anderen Knoten angepasst. Wenn also bei der Lösung des Teilziels A 3 Punkte des zusätzlichen Teils verbraucht wurden, dann wird für das Teilziel B der zusätzliche Anteil um 3 Punkte reduziert. Die Abbildung 4 soll diesen Sachverhalt verdeutlichen.

In der 1. Zeile waren noch 7 Schritte erlaubt. In der zweiten Zeile waren dann ursprünglich noch 6 Schritte verblieben, wovon je 1 Schritt pro Teilziel garantiert ist und 3 zusätzliche Schritte insgesamt vorhanden sind. Zur Lösung des linken Teilziels wurden jedoch zwei Schritte benötigt, wodurch ein zusätzlicher Schritt verbraucht wurde und somit stehen zur Verfolgung der anderen beiden Teilziele nur noch der jeweils garantiere Schritt und zwei zusätzliche zur Verfügung.

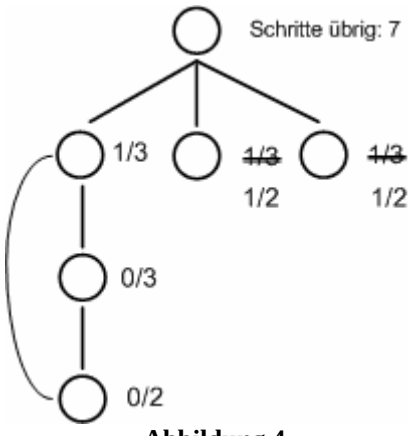


Abbildung 4

5. Teilziel Wahl im Konnektions-Tableau

Trotz all dieser Grenzen gibt es immer noch einen gewissen Zufallsfaktor im System, nämlich was wählt das System als nächstes zu bearbeitendes Teilziel und welche Literale werden infolge dessen angehangen. Dieses Problem kann man als eine unabhängige Funktion betrachten und so losgelöst von einem spezifischen System untersuchen. Im Wesentlichen gibt es zwei Lösungsansätze die *Fewest Solution* und die *Depth First* Methode. Bei der *Fewest Solution* wird immer versucht das Teilziel zu verfolgen, welches mit den wenigsten Aufspaltungen vorangetrieben werden kann. Bei *Depth First* dagegen wird ein Teilziel immer erst komplett abgearbeitet, bevorzugt wird hierbei natürlich jenes Teilziel welches am weitesten Weg ist von der Wurzel. Die Auswahl des Teilziels kann sowohl dynamisch als auch statisch erfolgen. Im statischen Fall wird vorher die Strategie festgelegt, wie beispielsweise das immer das ganz linke (auf Englisch: left-most) ausgewählt wird, während bei der dynamischem Auswahl ein der Situation angepasstes Teilziel auswählt wird.
Wurde ein Teilziel gewählt, so ergeben sich eine Menge von möglichen Unifikationspartnern und im Falle einer Sackgasse kann man zu diesem so genannten *Choice-Point* zurückkehren, um einen anderen Unifikationspartner zu wählen. Diese Praktik ist beispielsweise aus Prolog bekannt. Im der Praxis hat sich die *Fewest Solution* Methode mehr bewährt, da mit steigender Baumtiefe alle Unifikationen mitbetrachtet werden müssen. Folgendes konstruiertes Beispiel soll die Problematik zwischen den beiden Methoden verdeutlichen.

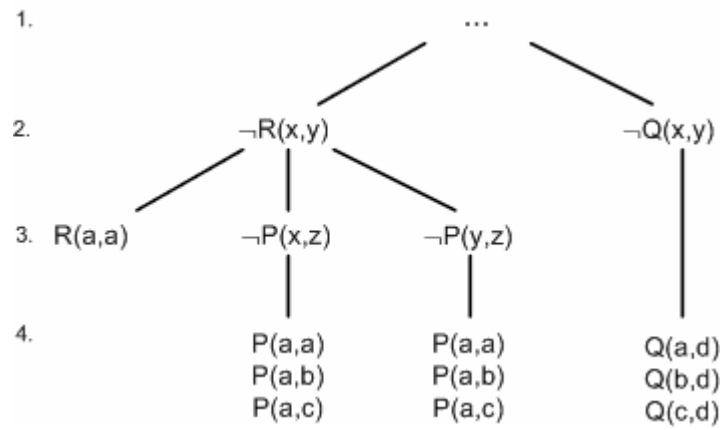


Abbildung 5

Bemerkung: Die je 3 Lierale in der 4. Zeile sollen lediglich als Auswahlmöglichkeit verstanden werden und wurden nur zur bessern Orientierung gleichzeitig hingeschrieben. Das System wählt davon natürlich nur einen aus.

Wird nach der *Depth First* Methode verbunden mit der Strategie des *left-most* Verfahrens vorgegangen, so werden zuerst die 3 Teilziele von *¬R(x,y)* abgearbeitet. Dies geschieht laut der gewählten Strategie von links nach rechts. Durch die Konnektion von *R(a,a)* mit *¬R(x,y)* werden sowohl x als auch y mit a unifiziert. Das Teilziel *¬P(x,z)* kann nun entweder mit *P(a,a), P(a,b)* oder *P(a,c)* konnektiert werden, je nach dem ergibt sich als Unifikation für z entweder a, b oder c. Das gleiche gilt für das letzte Teileziel *¬P(y,z)*. Hier wird z auch entweder mit a, b oder c belegt. Wird nun versucht das Teilziel *¬Q(x,y)* zu lösen, stehen die 3 Möglichkeiten *Q(a,d), Q(b,d), Q(c,d)* zur Verfügung. Das System durchläuft alle drei Möglichkeiten, findet aber keine passende Unifikation. Denn wie wir wissen ist y schon mit a unifiziert worden und darf deswegen nicht mit d unifiziert werden. Nachdem alle 3 Möglichkeiten erfolglos durchlaufen wurden, kehrt das System zum letzten *Choice-Point* zurück. Das ist *¬P(y,z)* und wählt hier eine der zwei verbleibenden Möglichkeiten, als

nächstes gelangt das System wieder zu $\neg Q(x,y)$ und durchläuft wieder erfolglos die drei Varianten. Dies setzt sich nun so fort und nachdem beim *Choice-Point* von $\neg P(y,z)$ auch die letzte Möglichkeit fehlgeschlagen ist, wird noch eins weiter bis zum Choice-Point von $\neg P(x,z)$ zurückgegangen. Von dort aus wird mit einer anderen Belegung fortgefahren. Insgesamt werden also 27 Belegungen des Komplements vom Teilziels $\neg Q(x,y)$ erfolglos ausprobiert.
Wird dagegen die *Fewest Solution* verwendet, so wird zuerst versucht das Teilziel $\neg Q(x,y)$ zu lösen. Denn hier ergibt sich nur 1 Ast im Gegensatz zu den 3 Ästen des Teilziels $\neg R(x,y)$. Hier gibt es nun die Möglichkeit $\neg Q(x,y)$ entweder mit $Q(a,d)$, $Q(b,d)$ oder $Q(c,d)$ zu unifizieren, je nach dem muss x entweder mit a, b oder c unifiziert werden, aber y immer mit d. Macht man sich dann an die Lösung von $\neg R(x,y)$ so scheitert man gleich am ersten Ast $R(a,a)$, da y schon mit d unifiziert wurde. Wenn nun zum *Choice-Point* von $\neg Q(x,y)$ zurückgesprungen wird, gibt es noch zwei weitere Möglichkeiten. Doch leider sind beide erfolglos, so dass nach 3 Unifikationen für $\neg Q(x,y)$ und dem anschließenden scheitern an $R(a,a)$ mit deutlich weniger Gesamtschritten abgebrochen wird, als im Vergleich zu der *Depth First* Methode.
Eine weitere Optimierung besteht darin, dass wenn mehrere Teilziele mit gleicher Länge (gemäß *Fewest Solution*) zur Auswahl stehen, immer das zu wählen, bei dem der Unifikationsparnter an den Choice-Points die geringste Klausellänge hat. Gerät man dann in eine Sackgasse hält man nicht wie beim normalen Backtracking an seinem aktuellen Teilziel fest, sondern sucht unter allen Teilzielen (gemäß *Fewest Solution*) wieder den Unifikationsparnter mit der kleinsten Länge. Es werden also quasi mehrere Choice-Points simultan beobachtet und aus ihnen die beste Wahl herausgegriffen. Unter gewissen Umständen hat diese Methode aber auch seine Nachteile. Denn kann ein Teilziel insgesamt betrachtet nicht gelöst werden, so verzögert das hin- und herspringen zwischen den Teilzielen die Einsicht, dass der ganze Beweis nicht lösbar ist und das System würde möglicherweise länger laufen, als beim einfachen Backtracking.

6. Fazit

In diesem Abschnitt wurden die Grundlagen zum Umgang mit Konnektions-Tableaus geschaffen, dabei wurde speziell auf deren Aufbau, verbunden mit der anschließenden Beweissuche bis hin zur passenden Teilziel Wahl eingegangen und so der Grundstein für das nächste Kapitel *„Verfeinerungen bei Konnektions-Tableaus“* gelegt.

Quelle:

Handbook of Automated Reasoning
Elsevier 2001 – Alan Robinson & Andrei Voronkov
Kapitel 28: Model Elimination and Connection Tableau Procedures