

Wintersemester 2006/07

Projekt: Reasoning Services: Ein Tableau-Beweiser für
Beschreibungslogiken

Carola Eschenbach
Özgür Özçep

*Effiziente Expansion durch
Redundanzvermeidung in einem
beschreibungslogischen Tableau-Beweiser*

Baccalaureatsarbeit

Arved Solth
Matrikelnummer 5714919

Inhaltsverzeichnis

Einleitung.....	3
Kapitel 1 Ablauf des Projektes.....	3
Kapitel 2 Theoretische Grundlagen.....	3
2.1 Beschreibungslogiken	3
2.2 Tableaubeweiser.....	7
Kapitel 3 Implementierung des beschreibungslogischen Tableaubeweisers.....	9
Kapitel 4 Komplexitätsverminderung durch Reduzierung von redundanten Expansionen.....	11
Kapitel 5 Optimierungen.....	15
5.1 Semantic Branching.....	15
5.2 Simplification.....	18
5.3 Weitere Optimierungen.....	20
Literatur.....	21

Einleitung

Das Projekt diente dazu, ein bereits bestehendes System eines aussagenlogischen Tableaubeweislers zu einem beschreibungslogischen Beweiser zu erweitern. Dabei sollte die Struktur soweit wie möglich beibehalten und nur in die Funktionalität betreffenden und erweiternden Fällen angepasst beziehungsweise verändert werden. Außerdem wurden verschiedene Optimierungsverfahren implementiert.

Kapitel 1 Ablauf des Projektes

In Kapitel 2 werden die theoretischen Grundlagen bezüglich beschreibungslogischer Sprachen und Tableaubeweiser als Beweisverfahren erläutert. Dabei wurden vor allem die Texte „Basic Description Logics“ [1] von Franz Baader und Werner Nutt, „Appendix 1. Description Logic Terminology“[2] von Baader sowie die Einführung zu Tableaubeweisern von Melvin Fitting aus dem Buch „Handbook of Tableau Methods“[3] von M. D'Agostino, D.M. Gabbay, R. Hähnle und J. Possega benutzt.

Diese Grundlagen wurden daraufhin in den aussagenlogischen Tableaubeweiser integriert.

Im letzten Abschnitt des Projektes wurden mehrere Optimierungen zur Performance-Verbesserung in den Beweiser eingebaut, deren Grundlagen vor allem aus dem Artikel „Optimising Description Logics“ [4] von Horrocks und Patel-Schneider stammen.

Kapitel 2 Theoretische Grundlagen

Im Folgenden möchte ich auf ein paar Grundlagen zu Beschreibungslogiken und Tableaubeweisern eingehen, um die darauf folgenden Erläuterungen zur Implementation unseres Systems verständlich zu machen.

2.1 Beschreibungslogiken

Franz Baader und Werner Nutt erklären in „Basic Description Logics“[1], wie Beschreibungslogiken aufgebaut sind und wie sie sich von anderen logischen Sprachen, wie zum Beispiel Aussagenlogik oder Prädikatenlogik unterscheiden.

Ein beschreibungslogisches System besteht aus einer Wissensbasis und einer Menge von Reasoning Services, die dem Benutzer verschiedene Dienste zum Arbeiten mit der Wissensbasis an die Hand geben.

Wissensbasis

Eine beschreibungslogische Wissensbasis besteht aus einer Terminologie, der so genannten T-Box, und einer Menge von Zusicherungen, der A-Box, die Aussagen zu konkreten Individuen machen.

In der Terminologie stehen Konzepte und Rollen, durch die Eigenschaften von Individuen der betrachteten Domäne beschrieben werden können. Als Beispiel führen Baader und Nutt eine Familien-Terminologie ein, in der Konzepte entsprechend familiärer Beschreibungen wie 'Mutter', 'Vater' oder 'Großmutter' eingetragen sind. Durch Rollen werden Beziehungen zwischen Individuen ausgedrückt, wie zum Beispiel 'hatKind' oder 'istVerheiratetMit'. Konzepte und Rollen können sowohl atomar als auch komplex vorkommen. Komplexe Konzepte und Rollen werden durch einen

Namen und eine Beschreibung definiert. Z.B. kann durch das Konzept 'Elternteil' anschaulich gemacht werden, wie Konzeptbildungsoperatoren dazu genutzt werden, komplexe Konzepte (und ggf. Rollen), bestehend aus einem Namen und einer Beschreibung zu bilden. Für 'Elternteil' könnte z.B. der disjunktive Operator ' \cup ' ('Vereinigung') eine Auswahl zwischen 'Vater' und 'Mutter' als 'Elternteil' darstellen, also:

$$\text{Elternteil} = \text{Vater} \cup \text{Mutter}$$

In diesem Fall ist 'Elternteil' der Konzeptname und die Disjunktion ' $\text{Vater} \cup \text{Mutter}$ ' die Konzeptbeschreibung. Auch 'Mutter' könnte man näher, zum Beispiel mit dem Konzeptbildungsoperator ' \cap ' ('Durchschnitt') als eine Frau beschreiben, die zusätzlich noch mit einem Individuum, das selber eine Person ist, in der Rollenbeziehung 'hatKind' steht, also

$$\text{Mutter} = \exists \text{Frau} \cap \text{hatKind.Person}$$

Da jedes Element der Domäne, das durch das Konzept 'Mutter' beschrieben wird, auch gleichzeitig durch das Konzept 'Frau' beschrieben wird, sagt man auch 'Das Konzept Frau subsumiert das Konzept Mutter':

$$\text{Mutter} \subseteq \text{Frau}$$

Subsumtionshierarchien aufzubauen gehört dabei auch zu den klassischen Aufgaben von Beschreibungslogiken.

Rollen dienen, wie an dem Beispiel oben bereits gesehen, dazu, Individuen in Beziehung zueinander zu bringen. So kann zum Beispiel 'istVerheiratetMit(X,Y)' bedeuten, dass die Individuen X und Y ein Ehepaar sind.

Nach der Menge der zur Verfügung stehenden Konzeptbildungsoperatoren wird auch die Mächtigkeit einer Beschreibungslogik definiert. Die folgende kontextfreie Grammatik listet die Operatoren auf, die die Grundlage für alle Beschreibungslogiken und in der einfachsten Beschreibungslogik AL verfügbar sind. Sei A ein atomares sowie C, D und E beliebige Konzepte.

<i>Universelles Konzept:</i>	$C \rightarrow \top$
<i>Unerfüllbares Konzept:</i>	$C \rightarrow \perp$
<i>atomares Konzept:</i>	$C \rightarrow A$
<i>atomare Negation:</i>	$C \rightarrow \neg A$
<i>Durchschnitt:</i>	$C \rightarrow D \cap E$
<i>Werte-Restriktion:</i>	$C \rightarrow \forall R.D$
<i>eingeschränkte Existenz-Restriktion:</i>	$C \rightarrow \exists R.\top$

Diese Standardsprache kann um weitere Operatoren wie Disjunktion [\cup], qualifizierte Existenzrestriktionen [\exists], Anzahlrestriktionen [\geq] und nicht-atomare Negation [\neg] erweitert werden.

<i>Vereinigung:</i>	$C \rightarrow D \cup E$
<i>qualifizierte Existenz-Restriktionen:</i>	$C \rightarrow \exists R.D$
<i>\geq-Anzahlrestriktionen:</i>	$C \rightarrow \geq nR$
<i>\leq-Anzahlrestriktionen:</i>	$C \rightarrow \leq nR$
<i>nicht-atomare Negation:</i>	$C \rightarrow \neg D$

Entsprechend bezeichnen Baader und Nutt die dabei entstehenden Beschreibungslogiken als AL mit den Erweiterungen $[U],[E],[N]$ und $[C]$, je nachdem, welche Operatoren in der Sprache enthalten sind. Wir haben uns im Projekt für die Sprache $AL[U][E]$ entschieden, also die Standardsprache zuzüglich Vereinigung und qualifizierter Existenzrestriktionen. Dabei gilt, dass $AL[U][E]$ mit der Sprache $AL[C]$ äquivalent ist.

Die Semantik der Operatoren kann durch eine Interpretation I als eine Abbildung von der Menge der Konzepte in die betrachtete Domäne Δ erklärt werden. Mit dieser wird jedem Konzept-Symbol ein Element der Domäne und jeder Rolle eine binäre Relation auf dem Kartesischen Produkt $\Delta \times \Delta$ zugeordnet. Sei R eine atomare Rolle, A ein atomares und C und D beliebige Konzepte:

$$\begin{aligned} \top^I &= \Delta^I \\ \perp^I &= \emptyset \\ A^I &\subseteq \Delta^I \\ (\neg A)^I &= \Delta^I \setminus A^I \\ (C \cap D)^I &= C^I \cap D^I \\ (C \cup D)^I &= C^I \cup D^I \\ (\forall R.C)^I &= \{a \in \Delta^I \mid \forall b. (a,b) \in R^I \rightarrow b \in C^I\} \\ (\exists R.\top)^I &= \{a \in \Delta^I \mid \exists b. (a,b) \in R^I\} \\ (\exists R.C)^I &= \{a \in \Delta^I \mid \exists b. (a,b) \in R^I \wedge b \in C^I\} \\ (\geq nR)^I &= \{a \in \Delta^I \mid |\{b \mid (a,b) \in R^I\}| \geq n\} \\ (\leq nR)^I &= \{a \in \Delta^I \mid |\{b \mid (a,b) \in R^I\}| \leq n\} \end{aligned}$$

Die A-Box beinhaltet Aussagen über konkrete Individuen der betrachteten Domäne, also eine Art Weltbeschreibung. A-Box-Ausdrücke sind entweder Konzept-Ausdrücke der Form $C(a)$ oder Rollen-Ausdrücke der Form $R(a,b)$, wobei C ein Konzeptsymbol, R ein Rollensymbol und a und b Konstanten sind. Hierbei wird eine „open world assumption“ getroffen, das heißt, dass der Wahrheitsgehalt von Aussagen, die nicht in der A-Box eingetragen sind, nicht bekannt ist. Im Gegensatz dazu erwähnen die Autoren die „closed world assumption“ von Datenbanksystemen, in denen davon ausgegangen wird, dass alles, was nicht in der Datenbank eingetragen ist, auch nicht wahr ist, ein umfassendes Wissen also vorausgesetzt wird.

Reasoning Services

Weiterhin definieren die Autoren von [1] wichtige Dienste, so genannte Reasoning Services, die Hauptaufgaben für beschreibungslogische Wissensrepräsentationssysteme darstellen. Zu diesen gehören Erfüllbarkeits-, Subsumtions- und Äquivalenztests für Konzepte sowie Konsistenzprüfung, Instance-Checks und Retrievalanfragen für A-Box-Einträge. Wie Baader und Nutt zeigen, können viele dieser Aufgaben durch andere äquivalent ausgedrückt werden. Um zum Beispiel die Äquivalenz von zwei Konzepten C und D zu zeigen, kann auch bewiesen werden, dass Konzept C Konzept D subsumiert und umgekehrt:

$$C \equiv D \Leftrightarrow (D \subseteq C) \cap (C \subseteq D)$$

So kann es im praktischen Implementierungsfall genügen, nur einen der Reasoning Services umzusetzen und alle anderen auf diesen zurückzuführen. Folgende Reasoning Services werden in unserem Programm zur Verfügung gestellt:

T-Box: Erfüllbarkeits-, Subsumtions-, Äquivalenz und Disjunktheitstest von Konzepten

A-Box: Erfüllbarkeitstest, InstanceCheck, Retrieval-Anfrage für A-Box-Einträge

Sei T die Menge der Interpretation, Δ die betrachtete Domäne und A und B Konzepte. Dann können die Reasoning Services wie folgt beschrieben werden:

Erfüllbarkeitstest

Ein Konzept C ist erfüllbar, wenn es eine Interpretation I gibt, für die C^I nicht leer ist, d.h., die C auf ein Element aus Δ abbildet:

$$C \text{ erfüllbar}_T \Leftrightarrow \exists I \in T: C^I \neq \emptyset$$

Subsumtionstest

Der Subsumtionstest überprüft, ob ein Konzept C von einem anderen Konzept D subsumiert wird. Das bedeutet, dass jedes Individuum x , für das $C(x)$ gilt, auch $D(x)$ gilt. Folgende Notation wird verwendet:

$$C \subseteq_T D \Leftrightarrow \forall I \in T: C^I \subseteq D^I$$

Äquivalenz zwischen Konzepten

Zwei Konzepte C und D sind äquivalent, wenn sie unter allen Interpretationen gleich sind:

$$C \equiv_T D \Leftrightarrow \forall I \in T: C^I = D^I$$

Disjunktheit zwischen Konzepten

Zwei Konzepte C und D sind disjunkt, wenn ihr Durchschnitt in allen Interpretationen leer ist:

$$C, D \text{ disjunkt}_T \Leftrightarrow \forall I \in T: C^I \cap D^I = \emptyset$$

Erfüllbarkeitstest einer A-Box

Eine A-Box A ist erfüllbar, wenn es eine Interpretation gibt, für die alle Einträge von A wahr sind.

$$A \text{ erfüllbar} \Leftrightarrow \exists I \in T \forall C(x) \in A: C(x)^I = \text{true}$$

Instance-Check eines A-Box-Eintrags

Überprüft, ob für ein übergebenes Konzept C und ein Individuum x die Aussage $C(x)$ in der A-Box A steht bzw. aus dieser abgeleitet werden kann. Dies ist gleichbedeutend mit der Aussage, dass A vereinigt mit der Negation von $C(x)$ inkonsistent ist.

$$\text{InstanceOf}(C(x), A) \Leftrightarrow A \cup \{\neg C(x)\} \text{ inkonsistent}$$

Retrieval von A-Box-Einträgen

Alle Individuen x , für die ein Konzept C gilt, werden aus der A-Box A gesucht. Auch dieses Problem kann durch A-Box-Inkonsistenz ausgedrückt werden.

$$\text{Retrieval}(A, C) = \{x \mid A \cup \{\neg C(x)\} \text{ inkonsistent}\}$$

Um diese Services für beschreibungslogische Systeme zu implementieren, werden im Projekt Tableaubeweiser als Beweisverfahren verwendet, welche im nächsten Abschnitt etwas näher erklärt werden.

2.2 Tableaubeweiser

Fitting beschreibt in seiner Einführung zu Tableaubeweisern[3], wie diese über die Zeit als intuitive und oft genutzte Werkzeuge für das Beweisen von logischen Formeln entstanden sind. Das Tableau ist nach seinen Ausführungen aus den grundlegenden Arbeiten von Gentzen, Beth, Hintikka, Lys und schließlich Smullyan hervorgegangen.

Tableaubeweiser sind eine weit verbreitete Art, Formeln verschiedener logischer Sprachen auf Erfüllbarkeit zu überprüfen. Es gibt Tableaubeweiser mit entsprechenden Regeln für Sprachen von klassischer Aussagenlogik über modallogische Systeme bis hin zu solchen für vollständige Prädikatenlogik. Die Unentscheidbarkeit der Prädikatenlogik macht dadurch auch das Tableaubeweiser-Verfahren, das ansonsten ein Entscheidungsverfahren für das Erfüllbarkeitsproblem darstellt, im prädikatenlogischen Fall zu einem Semientscheidungsverfahren.

Allgemein sind Tableaus Beweisverfahren, die die Gültigkeit von Formeln durch die Unerfüllbarkeit ihrer Negation zeigen. Dazu wird die negierte Formel als „Wurzel“ des Tableaus genommen und durch bestimmte Regeln solange neue Einträge aus den bestehenden expandiert und gegebenenfalls Zweige hinzugefügt, bis keine weiteren Regeln mehr anwendbar sind oder alle Zweige abgeschlossen sind. Ein Zweig A ist abgeschlossen, wenn eine der folgenden Abschlussbedingungen auf ihn zutrifft:

1. $\{\perp(x)\} \subseteq A$ für ein Individuum x
2. $\{C(x), \neg C(x)\} \subseteq A$ für ein Konzept C und ein Individuum x
3. $\{\leq n R(x)\} \cup \{R(x, y_i) \mid 1 \leq i \leq n+1\} \cup \{y_i \neq y_j \mid 1 \leq i < j \leq n+1\} \subseteq A$ für ein Rollennamen R , eine positive Zahl n und Individuen x, y_1, \dots, y_{n+1}

Sind alle Zweige abgeschlossen, ist auch das Tableau abgeschlossen und die anfängliche Formel damit als gültig bewiesen, da ihre Negation als unerfüllbar gezeigt wurde. Ist ein Zweig dagegen nicht abgeschlossen, bedeutet das, dass es eine Interpretation für die negierte Formel gibt, in der diese zu „wahr“ und die ursprüngliche Formel damit zu „falsch“ ausgewertet wird, wodurch sie nicht mehr gültig ist. Fitting bezeichnet dabei das Tableau auch als generalisierte disjunktive Normalform, wobei die einzelnen Zweige als Disjunktionsglieder und die Einträge in den Zweigen als die Konjunktionsglieder innerhalb eines Disjunktionsgliedes angesehen werden können.

Eine besondere Art von Tableau, die Fitting hervorhebt und die wir auch in unserem Projekt verwendet haben, ist das markierte Tableau. Bei diesem bekommen Formeln ein Label bzw. eine Markierung, die kennzeichnet, ob eine Formel in der aktuellen Interpretation, das heißt in dem aktuellen Zweig des Tableau, als wahr oder falsch interpretiert vorkommt. Durch diese Markierung ist es besonders einfach, einen Abschluss festzustellen, da ein Zweig bereits zum Abschluss geführt werden kann, wenn eine potentiell komplexe Formel mit zwei verschiedenen Labels im Zweig vorkommt. Dies würde offensichtlich einer Inkonsistenz in der Interpretation entsprechen und einen Abschluss des Zweigen bedeuten. Die Expansionsregeln für markierte Tableau:

<i>Negation:</i>	$\frac{T : \neg X}{F : X}$	$\frac{F : \neg X}{T : X}$
<i>Konjunktion:</i>	$\frac{T : X \cap Y}{T : X}$ $T : Y$	$\frac{F : X \cap Y}{F : X \mid F : Y}$
<i>Disjunktion:</i>	$\frac{T : X \cup Y}{T : X \mid T : Y}$	$\frac{F : X \cup Y}{F : X}$ $F : Y$

<i>Werte-Restriktion:</i>	$\frac{T:\forall R.C(x)}{T:R(x,y)} \quad T:C(y)$	$\frac{F:\forall R.C(x)}{F:C(y)} \quad T:R(x,y)$
<i>Existenz-Restriktion:</i>	$\frac{T:\exists R.C(x)}{T:C(y)} \quad T:R(x,y)$	$\frac{F:\exists R.C(x)}{T:R(x,y)} \quad F:C(y)$

Da wir in unserem Projekt nicht die Gültigkeit, sondern die Erfüllbarkeit von beschreibungslogischen Ausdrücken überprüfen wollten, wird in unserem Tableau die ursprüngliche Formel und nicht ihre Negation als Ausgangspunkt gewählt. Wenn in einem solchen Tableau auch nach vollständiger Expansion noch Zweige offen sind, bedeutet das, dass es eine Interpretation gibt, die die Ursprungsformel wahr macht und diese damit erfüllbar ist.

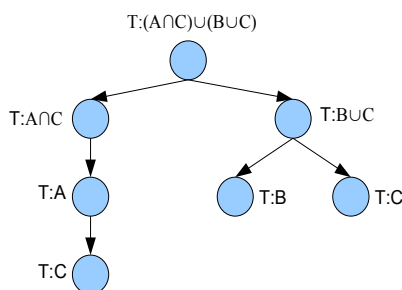
Fitting geht bei seiner Beschreibung besonders auf die Arbeit von Smullyan ein, der für Tableaus die Uniforme Notation eingeführt hat. Hierbei wird nicht mehr nach einzelnen Formeln mit unterschiedlichen Markierungen unterschieden, auf die jeweils bestimmte Regeln angewendet werden können, sondern Gruppen von Formel-Markierungs-Paaren werden zusammengefasst, die auf die gleiche Art expandiert werden. Dabei wird im aussagenlogischen Fall nur nach 'verzweigend' und 'nicht-verzweigend' unterschieden. Eine mit 'false' markierte Konjunktion, z.B. $F:A \wedge B$, führt genauso zu einer Verzweigung wie eine mit 'true' markierte Disjunktion, wie $T:C \vee D$. Diese Formeln werden von Smullyan β -Formeln genannt. Als α -Formeln bezeichnet er nicht-verzweigende Formeln, also mit 'true' markierte Konjunktionen und mit 'false' markierte Disjunktionen.

$$\alpha\text{-Formel: } \frac{\alpha}{\alpha_1 \alpha_2} \quad \beta\text{-Formel: } \frac{\beta}{\beta_1 | \beta_2}$$

α - und β -Formeln beschreiben unter Hinzunahme einer Regel für die Negation, die nur die Markierung der Formel umkehrt, bereits den kompletten Sprachumfang der klassischen Aussagenlogik. Für komplexere Logiken wie Modallogiken werden weitere Formeln in Gruppen eingeteilt und entsprechende Bezeichner wie ν - und π -Formeln erstellt.

Diese kompakte Darstellungsform ist eine Möglichkeit, Formeln einheitlich zu repräsentieren. In unserem Projekt haben wir sie jedoch nicht explizit verwendet.

Folgende Figur stellt ein markiertes Tableau für einen einfachen aussagenlogischen Ausdruck dar, und soll nur dazu dienen, die Grundlagen zu veranschaulichen:



Markiertes aussagenlogisches Tableau

Kapitel 3 Implementierung des beschreibungslogischen Tableaubeweisers

Ausgangspunkt ist der oben bereits erwähnte aussagenlogische Tableau-Beweiser. Der Beweiser benutzt ein Tableau, das grundsätzlich aus einer Liste von Zweigen (in unserem Projekt *TBranch*) besteht, die wiederum aus einer Liste von Einträgen (*TEntry*) aufgebaut sind. Ein Eintrag setzt sich zusammen aus einer Markierung (*TLabel*) und einer aussagenlogischen Formel (*Expression*). Als einzigen Eintrag bekommt das Tableau am Anfang die zu überprüfende Formel. Um diesen Eintrag zu expandieren, benutzt der Beweiser Expansionsregeln (*ExpansionRule*) und erstellt mit deren Hilfe Expansionen (*TExpansion*). Diese erzeugen neue Zweige, die in die Liste der Tableau-Zweige aufgenommen werden. Da das Tableau als Liste von Zweigen und nicht als Baumstruktur realisiert wurde, werden alle Zweige vom Blatt bis zur Wurzel (der ursprünglichen Formel) komplett gespeichert, und es entstehen bei Verzweigungen gewisse Redundanzen, da Einträge, die in mehreren Zweigen liegen, in jedem einzelnen dieser Zweige gespeichert werden.

Unter anderem müssen folgende Punkte angepasst werden:

- Erweiterung des Provers zu einem beschreibungslogischen Beweiser
- Anpassen der Tableau-Struktur, d.h.:
 - Überarbeitung der Darstellung von Zweigen und Einträgen im Tableau
 - Erzeugung einer neuen Datenstruktur, um beschreibungslogische Ausdrücke darzustellen
 - Erzeugung von neuen Operatoren für die verschiedenen, beschreibungslogischen Ausdrücke
- Erstellen neuer beschreibungslogischer Expansionsregeln für das Tableau

Ein Eintrag im ursprünglichen aussagenlogischen Tableau besteht aus einem Ausdruck und einer Markierung. In dem Ausdruck wird eine Formel in Form der Menge ihrer Teilformeln (*subformulae*) und des Hauptoperators (*mainOperator*) gespeichert. Die Markierung gibt an, ob die Formel an dieser Stelle im Tableau als 'true' oder 'false' interpretiert wird.

Um einen Eintrag für ein beschreibungslogisches Tableau nutzen zu können, müssen nun ein paar Ergänzungen vorgenommen werden. Dies ist nötig, da ein beschreibungslogischer Ausdruck immer eine Beschreibung eines Individuums ist. Daher bekommt der Eintrag zusätzlich zu der Markierung und dem Ausdruck eine Konstante, die für dieses Individuum steht. Um ein beschreibungslogisches Konzept oder eine Rolle auszudrücken, wird eine neue Art von Ausdruck (*DLExpression*) verwendet. Dieser hat als Felder eine Menge von beschreibungslogischen Teilausdrücken (*subExpressions*), einen Hauptoperator (*mainOperator*), eine ganzzahlige, positive Zahl für Anzahl-Restriktionen (*n*) sowie einen Namen (*name*). Letzterer wird nur bei atomaren Konzepten ohne Teilausdrücke vergeben. Entsprechend ist bei atomaren Konzepten auch das Array der Teilausdrücke *subExpressions* leer. Da Anzahl-Restriktionen noch nicht umgesetzt wurden, ist das *n* bisher noch ungenutzt.

Ein Zweig im beschreibungslogischen Tableau wird in verschiedene Listen von Einträgen unterteilt. Dabei wird unterschieden zwischen atomaren Einträgen (*atomicEntries*), Einträgen, die noch expandiert werden (*toBeExpanded*), bereits expandierten Einträgen (*alreadyExpanded*), atomaren Rolleneinträgen (*roleEntries*) und solchen, die im Zuge einer Werte-Restriktions-Regel-Anwendung eventuell noch einmal expandiert werden müssen (*potentiallyExpandedAgain*). Zusätzlich gibt es eine Liste aller Einträge (*entries*). Die verschiedenen Fälle für einen möglichen Abschluss eines Zweiges werden ebenfalls in dem Zweig überprüft. Möglich ist dabei der Test auf atomaren und auf nicht-atomaren Abschluss. Nicht-atomarer Abschluss bietet dabei den Vorteil zu

einer frühzeitigen Erkennung, da die Formeln ggf. nicht bis auf atomare Ebene expandiert werden müssen. Dies hat aber den Nachteil, dass wir mehr und längere Vergleiche haben.

Neue Zweige werden von Expansionen (*TExpansion*) erzeugt, die ihrerseits durch das Anwenden von Expansionsregeln (*DLExpansionRuleSimple*, *DLExpansionRuleBool*, *DLExpansionRuleQuantifier*) auf passende Einträge in einem Zweig entstehen. Expansionsregeln haben dabei Vorbedingungen, die erfüllt sein müssen, um sie auf einen Eintrag anwenden zu können. Beispielsweise muss zum Anwenden der verzweigenden Disjunktionsregel der *mainOperator* des Ausdruckes im aktuellen Eintrag ein '∪' und die Markierung ein 'true' sein oder, entsprechend den de'Morgan'schen Regeln und der uniformen Notation von Smullyan, ein '∩' mit einem 'false' als Markierung. Wird die Vorbedingung erfüllt, generiert die Expansionsregel eine Expansion, durch deren Anwendung auf den aktuellen Zweig dann ein oder mehrere neue Zweige entstehen.

Folgende Expansionsregeln wurden im beschreibungslogischen Tableau-Beweiser umgesetzt:

DLExpansionRuleSimple: anwendbar auf Negationen. Wandelt Negations-Symbole in Markierungen um.

$$\frac{T : \neg A(x)}{F : A(x)} \quad \frac{F : \neg A(x)}{T : A(x)}$$

DLExpansionRuleBool: anwendbar auf Ausdrücke mit den Hauptoperatoren '∩' oder '∪'. Generiert folgende Expansionen:

$$\frac{T : A_1 \cap \dots \cap A_n(x)}{T : A_1(x)} \quad \frac{F : A_1 \cap \dots \cap A_n(x)}{F : A_1(x) \mid \dots \mid F : A_n(x)}$$

$$\frac{T : A_1 \cup \dots \cup A_n(x)}{T : A_1(x) \mid \dots \mid T : A_n(x)} \quad \frac{F : A_1 \cup \dots \cup A_n(x)}{F : A_1(x)}$$

$$\frac{\dots}{T : A_n(x)} \quad \frac{\dots}{F : A_n(x)}$$

$$\frac{T : A_1 \cup \dots \cup A_n(x)}{F : A_1(x)}$$

$$\frac{F : A_1 \cap \dots \cap A_n(x)}{T : A_1(x)}$$

$$T : A_1(x) \mid \frac{F : A_1(x)}{T : A_2(x)} \mid \dots \mid \frac{\dots}{F : A_{n-1}(x)}$$

$$F : A_1(x) \mid \frac{T : A_1(x)}{F : A_2(x)} \mid \dots \mid \frac{\dots}{T : A_{n-1}(x)}$$

$$T : A_n(x) \quad F : A_n(x)$$

Hier dargestellt sind die Expansionsregeln für die generalisierten Disjunktionen und Konjunktionen, die beliebig viele Operanden unterstützen. Die letzten beiden Regeln präsentieren das Expansionsverhalten bei zusätzlich eingeschaltetem Semantic Branching, das im letzten Kapitel beschrieben wird.

DLExpansionRuleQuantifier: anwendbar auf Ausdrücke mit den Hauptoperatoren 'Werte-Restriktion' und 'Existenz-Restriktion'. Generiert folgende Expansionen:

$$\frac{T : \exists . R.C(x)}{T : C(y)} \quad \frac{F : \exists . R.C(x)}{T : R(x, y)} \quad \frac{T : \forall . R.C(x)}{T : R(x, y)} \quad \frac{F : \forall . R.C(x)}{F : C(y)}$$

$$T : R(x, y) \quad F : C(y) \quad T : C(y) \quad T : R(x, y)$$

Die mit 'true' markierte Werte-Restriktion und die mit 'false' markierte Existenz-Restriktion machen eine Besonderheit bei der Verarbeitung von Ausdrücken mit Restriktions-Operatoren deutlich. Hier muss aufgrund von mehr als einer Vorbedingung nicht nur der aktuelle Eintrag, sondern alle Rolleneinträge des aktuellen Zweig untersucht werden. Da dies mehr Aufwand als die Überprüfung der Vorbedingungen von booleschen Operatoren bedeutet, wurden die entsprechenden Regeln in eine eigene Klasse ausgelagert.

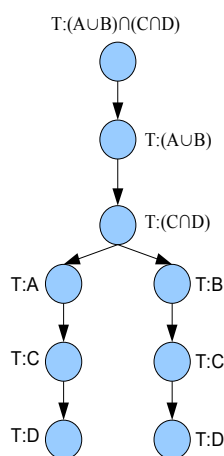
Kapitel 4 Komplexitätsverminderung durch Reduzierung von redundanten Expansionen

Verzweigungen

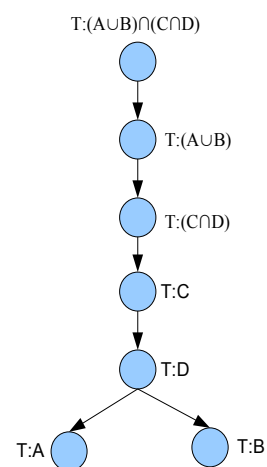
Verzweigungen, die durch die Anwendung von Expansionsregeln auf entsprechende Einträge entstehen, sind ein dominierender Faktor für die Komplexität eines resultierenden Tableaus. Daher ist es erstrebenswert, die Anzahl der Verzweigungen in einem Tableau soweit wie möglich zu minimieren, ohne dabei dessen Korrektheit zu beeinflussen. Bei den vorgestellten Tableau-Verfahren führen zwei Arten von Expansionsregeln zu Verzweigungen, die mit 'true' markierte Disjunktion (entsprechend die mit 'false' markierte Konjunktion) und die mit 'true' markierte höchstens-Anzahl-Restriktion ($\leq n$ R.C). Daher zielen Strategien zum Vermeiden von redundanten Verzweigungen besonders darauf ab, diese Arten von Expansionsregeln intelligent auszuführen. Um dies zu erreichen, gibt es mehrere Expansionsstrategien und -Optimierungen. Ein Optimierungsverfahren, das die Menge der nötigen Verzweigungen reduzieren kann, wird in Kapitel 6.2 Simplification beschrieben.

Prinzipiell versucht man, Verzweigungen bei der Expansion eines Tableau erst nach dem Ausführen aller anderen möglichen Expansionsregeln zu erzeugen, um die restliche 'Expansionsarbeit' nicht doppelt oder mehrfach in den einzelnen, resultierenden Zweigen durchführen zu müssen. Stehen verschiedene Einträge in einem Tableau bereit, die als nächstes expandiert werden können, wählt man einen solchen, auf den eine passende Regelanwendung keine Verzweigung generiert, bevor neue Zweige erzeugende Regeln angewendet werden. Dafür wird eine Priorisierbarkeit von Regeln angenommen, um nicht-verzweigende Regeln vor verzweigenden ausführen zu können.

Betrachtet man z.B. den unten dargestellten Eintrag $T:(A \cup B) \cap (C \cap D)$, stehen nach der Bearbeitung der ersten Konjunktion die zwei neuen Einträge $T:(A \cup B)$ und $T:(C \cap D)$ bereit, die als nächstes expandiert werden können. Wählt man als erstes die Disjunktion, ergeben sich zwei neue Zweige, jeweils einer für die Disjunkte $T:A$ und $T:B$. Die nun noch ausstehende Expansion der Konjunktion $T:(C \cap D)$ muss in beiden Zweigen vorgenommen werden, wodurch sich insgesamt 4 Expansionsschritte ergeben. Wird stattdessen die Konjunktion vor der Disjunktion bearbeitet, muss erstere nur einmal expandiert werden, da die Verzweigung erst im nächsten Schritt erzeugt wird. Also ergeben sich für diese Expansionsreihenfolge 3 Expansionsschritte.



Zuerst Disjunktion (4 Expansionen)



Zuerst Konjunktion (3 Expansionen)

Eine weitere Strategie zur Vermeidung von redundanten Expansionen besteht darin, bei Expansionsregeln vor ihrer Anwendung auf einen Eintrag zu prüfen, ob durch sie hinzugefügte Einträge oder Teile von ihnen in dem Tableau-Zweig eventuell schon existieren und die Expansion dadurch überflüssig wird.

In [1] werden für alle Expansionsregeln solche Überprüfungen in Form von zusätzlichen Vorbedingungen angegeben, wobei speziell die Regeln für Disjunktionen und qualifizierte Existenz-Restriktionen (entsprechend Konjunktionen und Werte-Restriktionen bei umgedrehter Markierung) effizienter erscheint. Wenn ein Eintrag mit einer mit 'true' markierten Disjunktion als Ausdruck gefunden wird, überprüft die passende Regel, ob eines der Disjunkte bereits mit 'true' markiert im untersuchten Zweig des Tableaus vorkommt. Ist dies der Fall, kann die Expansion der Disjunktion ausgelassen werden, da durch die verkürzte Auswertung bereits nach einem gefundenen, mit 'true' markierten Disjunkt feststeht, dass die Disjunktion zu 'true' ausgewertet wird und nicht mehr von ihren anderen Disjunkten abhängt.

Sei Θ der aktuelle Zweig und K_Θ die Menge der in Θ vorkommenden Konstanten. Dann können die entsprechenden Expansionsregeln für Disjunktion und Konjunktion wie folgt notiert werden:

$$\frac{\forall i, 1 \leq i \leq n: T : A_i(x) \notin \Theta \quad T : A_1 \cup \dots \cup A_n(x)}{T : A_1(x) | \dots | T : A_n(x)} \quad \frac{\forall i, 1 \leq i \leq n: F : A_i(x) \notin \Theta \quad F : A_1 \cap \dots \cap A_n(x)}{F : A_1(x) | \dots | F : A_n(x)}$$

Auch für die Expansionsregeln der qualifizierten Existenz-(und Werte-)Restriktionen kann durch eine solche Erweiterung der Vorbedingungen eine Expansion von Einträgen vermieden werden, falls sie für die weitere Bearbeitung des Zweiges keinen Unterschied macht. Für Restriktionen der Form $\exists R.C(x)$ wird dabei geprüft, ob im aktuellen Zweig eine Konstante z existiert, für die $R(x,z)$ und $C(z)$ bereits in den Zweig eingetragen sind. Dann muss keine neue Konstante erzeugt und keine Rollen- und Konzept-Einträge zu dem Zweig hinzugefügt werden.

Für Zweig Θ können diese erweiterten Regeln wie folgt angegeben werden:

$$\frac{\neg \exists y \in K_\Theta : T : R(x, y) \in \Theta \wedge T : C(y) \in \Theta \quad T : \exists . R.C(x)}{T : C(y) \quad T : R(x, y)} \quad \frac{\neg \exists y \in K_\Theta : F : C(y) \in \Theta \wedge T : R(x, y) \in \Theta \quad F : \forall . R.C(x)}{F : C(y) \quad T : R(x, y)}$$

Anzahl-Restriktionen und Rollen-Nachfolger

Eine weitere Strategie zur Optimierung von Tableau-Beweisern ist die Reduzierung von erzeugten Rollennachfolgern eines Individuums. Diese werden durch Existenz- und (in der Beschreibungslogik ALCN) \geq -Anzahl-Restriktionen generiert.

Baader und Nutt [1] beschreiben ein Tableau-Verfahren, bei dem der Erfüllbarkeitstest eines Konzeptes C durch Konsistenzprüfung einer ABox $A = \{C(x)\}$ vorgenommen wird. Diese ABox wird nach den im letzten Kapitel beschriebenen Expansionsregeln und zusätzlich durch Regeln für Anzahl-Restriktionen solange expandiert, bis keine Regel mehr angewendet werden kann. Falls ein Clash gefunden wurde, ist die ABox inkonsistent und C damit unerfüllbar, ansonsten ist C erfüllbar. Anzahl-Restriktionen werden nach folgenden Regeln expandiert:

$$T : (\leq nR)(x) \\ T : R(x, y_1)$$

\leq -Anzahl-Restriktionen:

$$\frac{\dots \quad T : R(x, y_{(n+1)})}{A[y_1/y_2] | \dots | A[y_n/y_{n+1}]}$$

wobei $A[y_i/y_j]$ ein neuer Zweig ist, der aus dem ursprünglichen Zweig durch Ersetzen aller Vorkommen von y_i durch y_j entsteht.

$$\frac{T : (\geq nR)(x)}{T : R(x, y_1)}$$

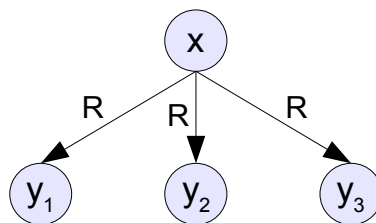
\geq -Anzahl-Restriktionen:

$$\begin{array}{c} \dots \\ T : R(x, y_n) \\ y_i \neq y_j, 1 \leq i < j \leq n \end{array}$$

Rollenbeziehungen zwischen Individuen haben die Struktur eines endlichen Baumes, dessen Knoten die durch Rollen in Beziehung zueinander stehenden Individuen und die Kanten zwischen ihnen die geltenden Rollen sind. Der Verzweigungsfaktor eines solchen Baumes ist durch die Summe von Existenz-Restriktions-Operatoren und den Zahlen aus \geq -Anzahl-Restriktionen begrenzt. Die Tiefe eines Rollenbaumes ist durch die maximale Rollentiefe des geprüften Konzeptes begrenzt. Baader und Nutt [1] erklären, dass um Regeln für Anzahl-Restriktionen erweiterte Tableau-Verfahren exponentiellen Zeit- und Platzbedarf haben können, wenn keine weiteren Optimierungsschritte gemacht werden.

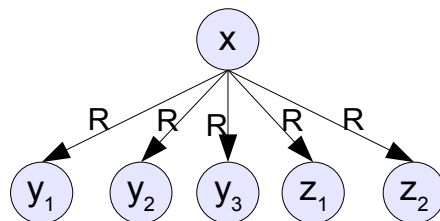
\leq -Anzahl-Restriktion können Individuen miteinander identifizieren, wenn mehr Rollennachfolger für eine Konstante existieren als es durch die Restriktion erlaubt ist. Daher wird davon ausgegangen, dass die unique-name-assumption (UNA) nicht gilt. Die unique-name-assumption besagt, dass jede unterschiedliche Konstante, die in einem Tableau vorkommt, auf ein unterschiedliches Element aus der betrachteten Domäne abgebildet wird. Stattdessen wird in einem Tableau durch \neq -Zusicherungen wie $a \neq b$ explizit angegeben, wenn zwei Konstanten verschieden sind und nicht miteinander identifiziert werden können.

Sei R eine Rolle sowie x , y_i und z_i Konstanten. Dann führt eine $\geq 3R$ -Anzahl-Restriktion $(\geq 3R)(x)$ zu dem Rollenbaum



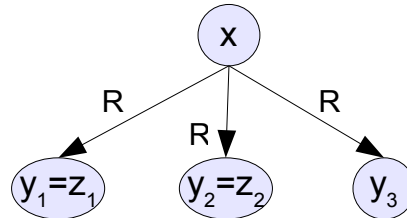
Rollenbaum für $(\geq 3R)(x)$

Befindet sich im gleichen Zweig zusätzlich eine $\leq n$ -Anzahl-Restriktion $(\leq 2R)(x)$, ergibt das einen Rollenbaum mit insgesamt 5 direkten Rollennachfolgern für x :



Rollenbaum für $(\geq 3R)(x), (\leq 2R)(x)$

In dieser Situation kann die $\leq n$ -Anzahl-Restriktion Individuen miteinander identifizieren, wenn deren Ungleichheit nicht explizit im Tableau durch \neq -Zusicherungen vermerkt ist. Das bedeutet, dass die Rollennachfolger, die durch die beiden Anzahl-Restriktionen für die Konstante x generiert wurden, nicht disjunkte Mengen sein müssen. Daher kann es sich bei den R-Nachfolgern der beiden Restriktionen um die gleichen Individuen handeln. Werden nun beispielsweise jeweils die Konstanten y_1 und z_1 sowie y_2 und z_2 miteinander identifiziert, ergibt sich folgender Rollenbaum:



Rollenbaum für $(\geq 3R)(x)$, $(\leq 2R)(x)$, nach Identifikation

Durch die Identifikation von Konstanten wird das ursprüngliche Tableau-Verfahren modifiziert:

1. wende alle anwendbaren \cup - und \cap -Regeln auf einen Zweig an und überprüfe, ob das Tableau durch Einträge der Art $\perp(x)$ oder $C(x)$ und $\neg C(x)$ abgeschlossen werden kann
2. erzeuge alle Rollen-Nachfolger einer Konstante durch Anwendung der passenden \exists - und \geq -Anzahl-Restriktionen
3. erzeuge Identifikationen zwischen den generierten Rollen-Nachfolgern durch Anwendung der passenden \leq -Anzahl-Restriktionen
4. behandle alle erzeugten Einträge auf die gleiche Art

Da dieser Algorithmus die nach der Identifikation verbleibenden Rollen-Nachfolger einzeln bearbeiten kann, muss immer nur ein Pfad des Rollenbaumes gespeichert werden. Dessen maximale Tiefe ist linear zu der maximalen Rollentiefe im untersuchten Ausdruck. Allerdings hängt der Platzbedarf des Algorithmus dabei auch von der Darstellung der Zahlen in den \geq -Anzahlrestriktionen ab. Wenn die Zahl zur Basis 1 notiert wird, entspricht die Größe der Darstellung der Größe der Zahl. Der Platzbedarf für die Darstellung wächst also linear mit der Zahl in der Anzahl-Restriktion an. Wird die Zahl jedoch in einer Notation zur Basis größer als 1, z.B. im Dezimalsystem dargestellt, wächst die Zahl und damit der Platzbedarf zum Speichern der Zahl exponentiell zu der Darstellung der Zahl an.

Wie Baader und Nutt erklären, reicht es, durch $\geq n$ -Anzahl-Restriktionen nur einen Rollen-Nachfolger eines Individuums einzuführen, anstatt n Nachfolger, wie bisher geschehen. Wenn bereits ein Rollen-Nachfolger y für ein Individuum x durch einen Rolleneintrag $R(x,y)$ in dem Zweig vorhanden ist, muss eine $\geq n$ -Anzahl-Restriktion $(\geq nR)(x)$ nicht mehr ausgeführt werden. Falls noch kein Nachfolger in dem Zweig vorhanden ist, wird durch die Anzahl-Restriktion nur noch ein Nachfolger generiert. Durch diese Modifikation wird das Darstellungsproblem der Zahlen in den \geq -Anzahl-Restriktionen gelöst und das von Baader und Nutt vorgestellte Verfahren hat nur noch polynomiellen Platzbedarf.

Um Inkonsistenzen durch konfligierende Anzahlrestriktionen zu erkennen, muss eine neue Art von Abschluß geprüft werden:

Seien n, m positive Zahlen mit $n < m$ und A eine ABox, dann:

$$\{(\leq nR)(x), (\geq mR)(x)\} \subseteq A \Rightarrow A \text{ abgeschlossen}$$

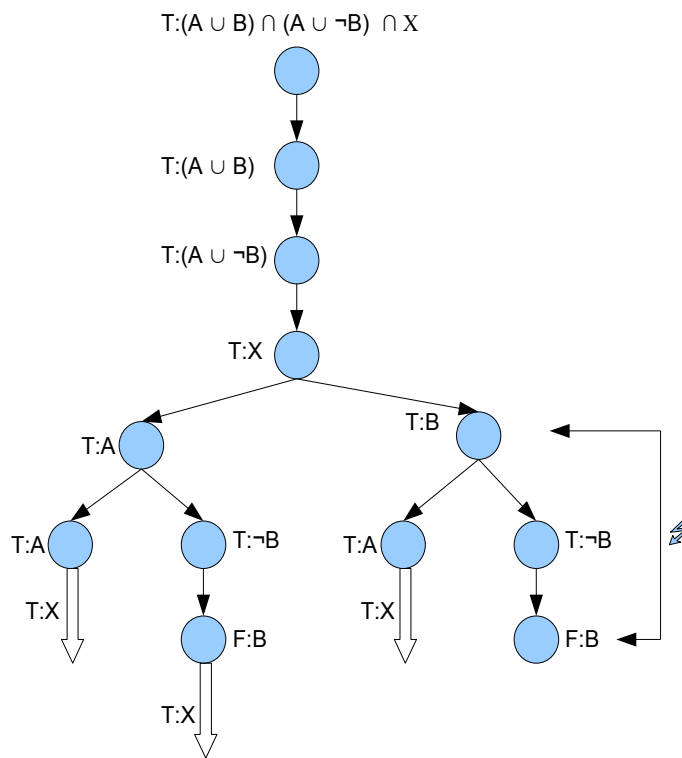
Kapitel 5 Optimierungen

In der abschließenden dritten Phase des Projektes wurden einige Optimierungstechniken für Tableaubeweiser aus dem Artikel „Optimising Description Logic Subsumption“[4] von Ian Horrocks umgesetzt. In diesem Kapitel gehe ich auf die Verfahren Semantic Branching und Simplification näher ein. Außerdem möchte ich kurz Dependency Directed Backtracking und die Normalisierung ansprechen, da auch diese Optimierungen im Projekt umgesetzt wurden. Auch Dependency Directed Backtracking vereinfacht die Tableau-Suche durch die Vermeidung von redundanten Expansionen. Normalisierung optimiert das Tableau-Verfahren durch einen anderen Ansatz, indem Umformungsregeln den zu prüfenden Ausdruck in eine leichter zu verarbeitende Normalform bringen, bevor das eigentlich Tableau-Verfahren beginnt.

5.1 Semantic Branching

Idee

Um die Grundidee vom Semantic Branching zu verdeutlichen, muss man sich zuerst die Funktionsweise des normalen Syntactic Branching in Erinnerung rufen. Bei diesem erzeugt eine Expansionsregel, die auf einen geeigneten Eintrag angewendet wird, genau die Anzahl von neuen Einträgen, die der verzweigende Ausdruck an Teilausdrücken hat. Dabei wird für jeden solchen Teilausdruck ein neuer Zweig generiert und der jeweilige Teilausdruck angehängt.



Syntactic Branching

Bei dieser Art der Verzweigung können redundante Expansionen vorkommen.

Betrachte man zum Beispiel den im obigen Tableau dargestellten Eintrag $T:(A \cup B) \cap (A \cup \neg B) \cap X$. Zuerst werden – wie oben beschrieben – die nicht verzweigenden vor den verzweigenden Regeln ausgeführt. Diese erzeugen die 3 neuen Einträge $T:(A \cup B)$, $T:(A \cup \neg B)$ und $T:X$ im Tableau. Wird nun die oben beschriebene Regel für syntaktisches Verzweigen auf eine der beiden Disjunktionen – z.B. $T:(A \cup B)$ - angewendet, entstehen zwei neue Zweige, mit den beiden Disjunkten A und B als

jeweils neue Einträge. Das gleiche geschieht nun auch mit der zweiten Disjunktion $T:(A \cup \neg B)$, sie erzeugt an beiden bestehenden Zweigen jeweils zwei neue Zweige, wieder mit den beiden Disjunkten als neuen Einträgen. Im aktuellen Expansionszustand kann nur einer der vier entstandenen Zweige abgeschlossen werden, derjenige, der durch die erste Verzweigung den Eintrag $T:B$ und durch die zweite Verzweigung den Eintrag $T:\neg B$ bekommen hat. Geht man nun weiterhin davon aus, dass der bisher nicht behandelte Teilausdruck X weitere aufwendige Expansionen benötigt, muss man diese nun in den übrigen 3 Zweigen separat durchführen. Führt die (teure) Expansion von X mit einem Eintrag weiter oben im Tableau, vor den Verzweigungen, zu einem Abschluss, leitet man den gleichen Clash 3mal her. Das Problem liegt darin, dass beim syntaktischen Verzweigen nicht zwangsläufig disjunkte Zweige entstehen, da für jedes Disjunkt jeder Verzweigung ein einzelner Zweig entsteht und nicht für die verschiedenen Möglichkeiten, ein Disjunkt zu interpretieren.

Um dies zu verhindern, kann Semantic Branching eingesetzt werden, dessen Ziel es ist, durch die Erzeugung von streng disjunkten Zweigen nur benötigte Expansionen durchzuführen und Redundanzen zu vermeiden. Dazu wählt der Algorithmus einen bislang noch nicht expandierten verzweigenden Eintrag aus und fügt pro Teilausdruck einen neuen Zweig an. In den ersten Zweig wird der erste Teilausdruck mit der ursprünglichen Markierung des Ausdruckes als neuer Eintrag übernommen, in den restlichen Zweigen wird dessen Markierung umgedreht. In den zweiten Zweig wird zusätzlich der zweite Teilausdruck als neuer Eintrag mit dem ursprünglichen Label übernommen. Auch dieser wird in die restlichen Zweige mit umgedrehten Label eingefügt. Auf diese Art werden alle Teilausdrücke in die erzeugten Zweige integriert. Da jeder dieser Teilausdrücke nur in einem Zweig mit der Markierung des ursprünglichen Ausdruckes auftaucht, und in allen anderen Zweigen ein anderes Label hat, sind alle Zweige streng disjunkt. Dadurch wird vermieden, dass im Folgenden auftretende Ausdrücke öfter als nötig expandiert werden, da alle Zweige eine eigene, zu den anderen unterschiedliche Semantik haben und dadurch redundante Expansionen vermieden werden.

Betrachtet man wieder das gleiche Beispiel wie oben, den Ausdruck $T:(A \cup B) \cap (A \cup \neg B) \cap X$, dann erkennt man, dass die vier entstehenden Zweige disjunkt sind, wenn sie durch Semantic Branching erzeugt werden. Die entstehenden Zweige erhalten durch die Verzweigungen folgende Einträge:

Zweig 1 : $\{T(A), T(A)\}$,

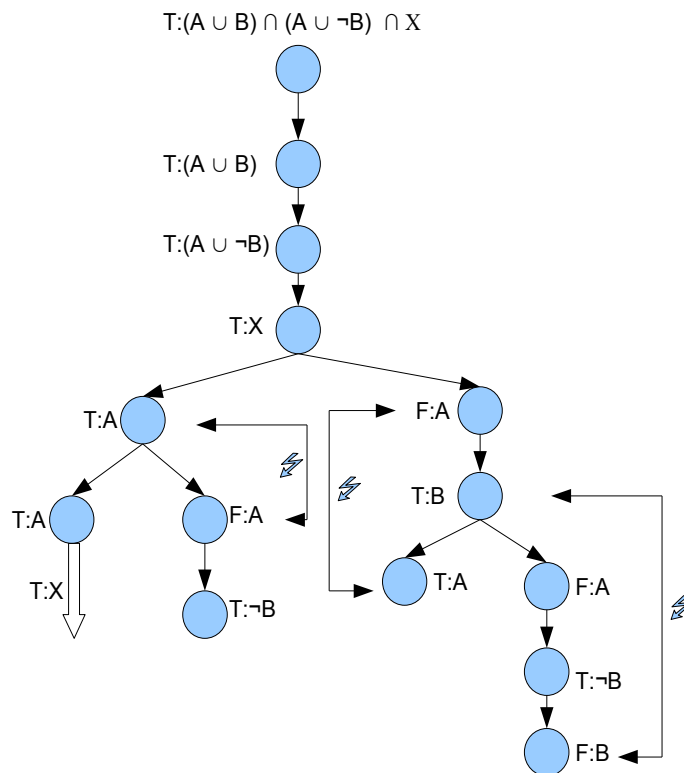
Zweig 2 : $\{T(A), F(A), T(B)\} \rightarrow \text{Abschluss}$

Zweig 3 : $\{F(A), T(B), T(A)\} \rightarrow \text{Abschluss}$

Zweig 4 : $\{F(A), T(B), F(A), F(B)\} \rightarrow \text{Abschluss}$

Da nach der Expansion der Verzweigungen durch Semantic Branching nur noch ein Zweig offen ist, muss der potentiell komplex zu expandierende Teilausdruck X nur noch einmal bearbeitet werden, anstatt dreimal, wie es beim Syntactic Branching der Fall war.

Semantic Branching stellt eine Art von Verzweigungsmöglichkeit dar, die alle möglichen Interpretationen eines Teilausdruckes unterscheidet, statt die verschiedenen Disjunkte einer Disjunktion. Dadurch ist garantiert, dass jeder einzelne Zweig einer anderen Interpretation entspricht und keine redundanten Verzweigungen generiert werden.



Semantic Branching

Umsetzung

Semantic Branching wurde von uns komplett in der Klasse „DLExpansionRuleBool“ implementiert, die für verzweigende Regeln zuständig ist. Dort kann gewählt werden, ob normales syntaktisches oder semantisches Verzweigen genutzt werden soll.

```

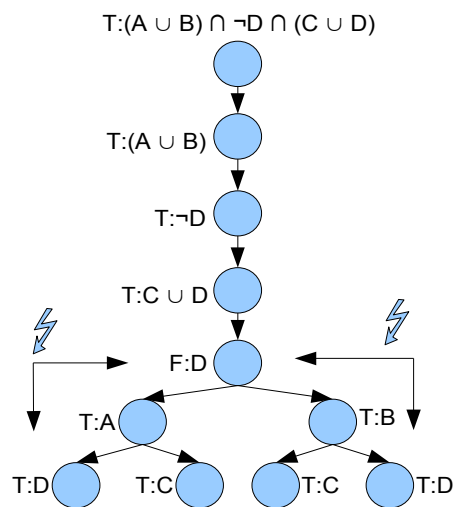
if(Eintrag verzweigend){
    for (alle Disjunkte a in der aktuellen Verzweigung){
        erzeuge einen neuen Zweig mit dem aktuellen Disjunkt mit dem ursprünglichen
        Label der Disjunktion und allen vorherigen Disjunkten mit dem ursprünglichen
        Label negiert;
    }
}
else {erzeuge neuen Zweig nach dem Syntactic Branching-Schema;}

```

5.2 Simplification

Idee

Die Simplification stellt ein Verfahren dar, das Verzweigungen vor ihrer Expansion vereinfacht. Wie schon erwähnt, sind Verzweigungen in Tableaubeweisern ein dominierender Faktor für deren Komplexität. Daher ist es wünschenswert, sie so stark wie möglich zu reduzieren. Um dies zu erreichen, geht das Verfahren vor Bearbeitung einer Verzweigung den aktuellen Zweig durch und durchsucht ihn nach Einträgen, die zum Streichen einzelner Disjunkte der aktuellen Verzweigung führen können. Gesucht werden Einträge, die den Teilformeln der Verzweigung entsprechen. Wird ein entsprechender Eintrag gefunden, werden die Label der Verzweigung und des einzelnen, gefundenen Eintrags verglichen. Falls diese nicht übereinstimmen, wird das entsprechende Disjunkt aus der Disjunktion entfernt. Durch dieses Verfahren können Verzweigungen vereinfacht oder sogar vermieden werden.



Ohne Simplification

Als Beispiel kann man ein Tableau mit dem Start-Eintrag $T:(A \cup B) \wedge \neg D \wedge (C \cup D)$ betrachten. Im ersten Schritt wird die Konjunktion expandiert. Diese erzeugt die Einträge $T:(A \cup B)$, $T:\neg D$ und $T:(C \cup D)$. Bei $T:\neg D$ wird die Negation in eine umgekehrte Markierung umgewandelt: $F:D$. Im nächsten Schritt wird die Verzweigung $T:(A \cup B)$ expandiert, wodurch zwei neue Zweige mit den Einträgen $T:A$ und $T:B$ generiert werden. $T:(C \cup D)$ erzeugt auf die gleiche Weise die neuen Einträge $T:C$ und $T:D$, jeweils doppelt für die beiden bereits bestehenden Zweige. Nun können 2 der 4 erzeugten Zweige durch die Einträge $T:D$ und $F:D$ abgeschlossen werden (siehe Darstellung oben).

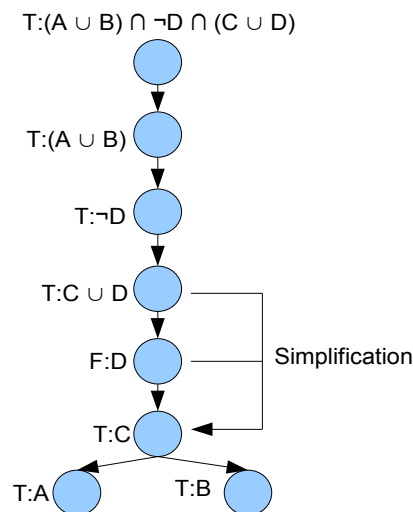
Um das Tableau zu vereinfachen, sucht Simplification vor der Expansion der ersten Verzweigung nach Vorkommen der Disjunkte einzelner Disjunktionen im behandelten Zweig. Der Eintrag $F:D$ wird gefunden. Da das Label dieses Eintrags (F) und das der Disjunktion $T:(C \cup D)$ verschieden sind, streicht der Algorithmus diesen Teilausdruck aus der Verzweigung. $T:(C \cup D)$ wird also durch $T:B$ ersetzt und die Verzweigung vermieden (siehe Abbildung unten).

Der Vorteil durch die vermiedene Erzeugung neuer Zweige besteht vor allem darin, dass weitere, nicht zu vereinfachende Verzweigungen sowie andere aufwändige Expansionen seltener durchgeführt werden müssen. In unserem konkreten Implementierungsfall ist dies zusätzlich von Vorteil, da neu erzeugte Zweige immer komplett vom Blatt bis zur Wurzel des Tableau-Baum und nicht nur bis zum letzten Verzweigungspunkt gespeichert werden. Hier führt das Vermeiden von redundanten Zweigen zu weiterem, gesparten Speicherplatz.

Umsetzung

Da die Simplification in dem kompletten aktuellen Zweig nach den Disjunkten der zur Zeit untersuchten Verzweigung sucht, braucht sie auch Zugriff auf den ganzen Zweig. Das unterscheidet sie von Semantic Branching, das nur lokal auf dem gerade zu expandierenden Eintrag eines Zweiges arbeitet.

Um den Zugriff auf den Zweig und seine Einträge zu gewährleisten, wird bereits beim Aufruf der Funktion der aktuelle Zweig als zusätzlicher Parameter übergeben. Dieser wird über die abstrakte Klasse „DLExpansionRule“ an „DLExpansionRuleBool“ weitergegeben, weil es sich um eine Expansionsregel handelt, die nur bei anstehenden Verzweigungen, also booleschen Operatoren, angewendet werden kann. In dieser Klasse wird nun die Suche nach dem Teilausdruck in dem Zweig gestartet. Wenn der Teilausdruck im Zweig gefunden wurde, wird er aus der Liste der Teilausdrücke des ursprünglichen Ausdruckes gelöscht und ein neuer Ausdruck generiert. Mit diesem wird nun eine Expansion generiert, die das Ergebnis der Simplification darstellt. In dem neuen, verzweigenden Ausdruck sind dann, falls die Simplification angewendet werden konnte, weniger Verzweigungsglieder und der Ausdruck wurde vereinfacht.



Mit Simplification

Die Vorgehensweise im Pseudo-Code:

```
Wähle zu bearbeitende Verzweigung aus Zweig aus;
for (alle Teilausdrücke a der Verzweigung){
    for (alle Ausdrücke b im aktuellen Zweig){
        if (a konfliktiert mit Negation von b) {
            speichere a in Liste zu löschender Teilausdrücke;
        }
    }
}
if (Liste zu löschender Teilausdrücke nicht leer){
    ziehe Elemente aus der Liste zu löschender Teilausdrücke von den Teilausdrücken
    von a ab;
    erzeuge neuen Eintrag aus resultierender Liste;
    hänge neuen Eintrag an den Zweig an;
}
```

5.3 Weitere Optimierungen

Dependency Directed Backtracking

Die Idee hinter Dependency Directed Backtracking ist es, alle Verzweigungen in einem Tableau mit Abhängigkeitspunkten (Dependency Points) zu kennzeichnen, um später zurückverfolgen zu können, an welchen Stellen Einträge, die zu einem Abschluss in einem Zweig geführt haben, entstanden sind.

Sobald ein Zweig abgeschlossen ist, springt der Algorithmus zum letzten Verzweigungspunkt zurück, durch den ein Eintrag hervorgegangen ist, der zum Abschluss beigetragen hat. Alle Verzweigungen dazwischen können übersprungen werden, da sie nicht zum Clash beigetragen haben. Die in diesen Verzweigungen erzeugten Zweige müssen daher nicht mehr untersucht werden, da sie durch die gleichen, weiter oben im Tableau liegenden Einträge zum Abschluss geführt werden können, wie der erste Zweig.

Hierdurch können große Mengen an Expansionen eingespart und die Tableau-Expansion entscheidend optimiert werden, wie Haarslav und Möller in [5] erläutern.

Normalisierung

Die Normalisierung basiert im Gegensatz zu den bisher vorgestellten Optimierungsverfahren nicht darauf, ein bereits bestehendes Tableau durch spezielle Regeln strategisch günstig zu expandieren, sondern die zu prüfende Formel bereits vor der ersten Expansion durch Umformungsregeln in eine Normalform zu bringen. Diese stellt eine einfacher zu verarbeitende Form der Formel dar und macht in günstigen Fällen eine anschließende Expansion komplett überflüssig. Durch die Umformungsregeln werden Disjunktionen und \exists -Operatoren aus der Formel entfernt und durch entsprechende Kombinationen aus Konjunktionen und Negationen bzw. \forall -Operatoren und Negationen ersetzt. Dadurch wird der Vergleichsaufwand zwischen Ausdrücken stark vereinfacht, da von einer einheitlichen Struktur ausgegangen werden kann.

Außerdem gibt es einen Satz lexikalischer Vereinfachungs-Regeln, die Wissen über die besondere Bedeutung von \top - und zu $\neg\top$ umgeformten \perp -Operatoren verwenden, um offensichtliche Kontradiktionen in Formeln schnell aufzudecken. Kommt beispielsweise ein $\neg\top$ in einer (generalisierten) Konjunktion vor, wird hier eine Art verkürzte Auswertung vorgenommen und die gesamte Konjunktion direkt zu einem $\neg\top$ umgeformt, ohne die anderen Konjunkte in dem Ausdruck zu betrachten.

Andere Optimierungsverfahren

Zusätzlich zu den hier kurz vorgestellten Optimierungen haben wir uns im Projekt noch die Verfahren Caching und Heuristik-gesteuerte Suche betrachtet. Diese wurden jedoch nicht in das Programm integriert. Das Heuristik-Verfahren wurde nicht übernommen, da der in dem Artikel beschriebene Performance-Gewinn die Gruppe nicht überzeugen konnte. Caching als Optimierung wurde von uns nicht implementiert, da dessen Umsetzung wohl den zeitlichen Rahmen des Projektes gesprengt hätte. Diese beiden Verfahren werden genau wie die oben erwähnten Optimierungs-Verfahren in [4] von Horrocks und Schneider näher erklärt.

Literatur

- [1] Franz Baader, Werner Nutt, Basic Description Logics. In F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi & P. Patel-Schneider (47-100), 2003
- [2] Franz Baader, Appendix1, Description Logic Terminology. In F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi & P. Patel-Schneider (485-495), 2003
- [3] Melvin Fitting, Introduction. In *Handbook of Tableau Methods* , M. D'Agostino, D.M. Gabbay, R. Hähnle, J. Possega (1-49), 1999
- [4] Ian Horrocks, Peter Patel.Schneider, Optimising Description Logic Subsumption. In *Journal of Logic and Computation* 9 (267-293), 1999
- [5] Volker Haarslav, Ralf Möller, *An Empirical Evaluation of Optimization Strategies for Abox Reasoning in Expressive Description Logics*, 1999