7 Einschränkung der Resolution

Der Resolutionskalkül hat durch die Verwendung allgemeinster Unifikatoren den Vorteil, daß die Verzweigungsrate im Suchraum endlich ist, d.h. für jede Klauselmenge gibt es nur endlich viele, und i.a. nicht allzu viele Möglichkeiten, Resolventen abzuleiten. Trotzdem gibt es noch nicht "den super-Resolutionsbeweiser, der alle machbaren Verifikations- und mathematischen Probleme löst. Einer Grund ist, daß immer noch sehr viele offensichtliche Redundanzen im Kalkül stecken, die die Suche erschweren.

7.1 Set-of-Support

Die am meisten benutzte und oft sehr wirkungsvolle Restriktionsstrategie ist *Set-of-Support.* Dazu teilt man die Klauselmenge auf in die Klauseln, die aus den Voraussetzungen entstanden sind, und die Klauseln, die von der negierten Behauptung stammen. Man geht davon aus, daß die Voraussetzungen nicht selbst bereits widersprüchlich sind und deshalb ein Widerspruch nur unter Beteiligung der Behauptung erzielt werden kann. Die Restriktion verbietet deshalb die Erzeugung von Resolventen zwischen zwei Voraussetzungs-klauseln.

Wenn im Beispiel $\{P,Q\},\{\neg P,Q\},\{P,\neg Q\},\{\neg P,\neg Q\}$ die Klausel $\{P,Q\}$ als Set-of-Support definiert ist, dann darf man nicht innerhalb der restlichen Klauselmenge resolvieren.

Im allgemeinen hat man die Axiome $A_1,\ldots A_n$ und einen Satz der Form $B_1 \wedge \ldots \wedge B_n \Rightarrow F$. Wenn B_i, F Literale sind, dann ist in der Klauselform nur $\neg F$ im set-of-support. Die Resolution mit SOS kann also nur damit anfangen zu schließen. Die Einschränkung ist am Anfang einer Beweissuche mittels Resolution sehr wirksam. Allerdings vergrößert sich die SOS-Menge nach einigen Resolutionsschritten, so daß die Wirkung nachläßt.

Etwas allgemeiner kann man die Klauselmenge, gegeben eine Struktur S in die Menge der Klauseln aufteilen, die von S erfüllt werden, und das Komplement. Ein Widerspruch kann nicht entstehen, wenn zwischen in S gültigen Klauseln resolviert wird.

7.2 UR-Resolution

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

$$C1\ \{\neg P(x,y), \neg P(y,z), P(x,z)\}$$

$$C2\ \{P(a,b)\}$$

$$C3\ \{P(b,c)\}$$

Damit sind u.a. folgende Resolutionsschritte möglich:

$$C1, 1 + C2 \vdash R1 = \{\neg P(b, z), P(a, z)\}$$

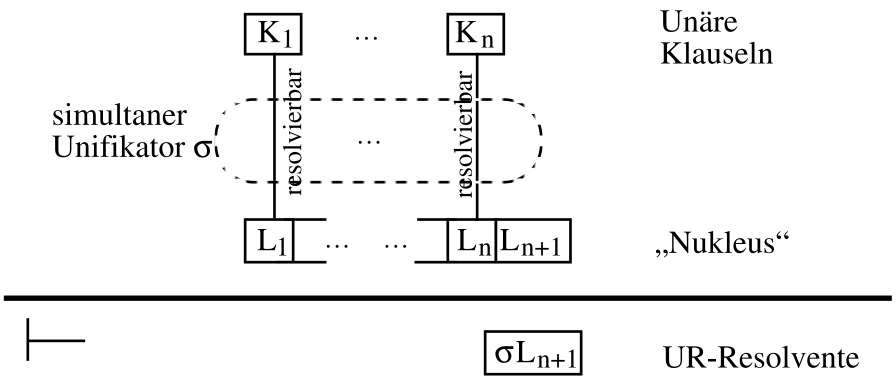
$$R1, 1 + C3 \vdash R2 = \{P(a, c)\}$$

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

$$C1, 2 + C3 \vdash \{R1' = \neg P(x, b), P(x, c)\}$$

$$R1, 1 + C2 \vdash R2 = \{P(a, c)\}$$

Die UR-Resolution faßt die beiden Schritte so zusammen, daß R2 unmittelbar abgeleitet wird und die Reihenfolge der Schritte keine Rolle mehr spielt. Das allgemeine Schema für UR-Resolution sieht in graphischer Darstellung folgendermaßen aus:



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

Die Wirkung der UR-Resolution ist nicht nur, daß von der Reihenfolge der n unären Resolutionsschritte abstrahiert wird. Obendrein werden die als Zwischenergebnisse anfallenden $n - 1$ Klauseln gar nicht erzeugt, im obigen Beispiel würden also $R1$ oder $R1'$ nicht in die Klauselmenge eingefügt. Da es dafür zunächst keinen Grund gibt, jedenfalls nicht mit irgendeiner gängigen Reduktionsregel, entspricht die UR-Resolution genaugenommen sogar einer ganz neuen Schlußregel, für die wieder dieselben Eigenschaften gezeigt werden müssen wie für die Resolutionsregel.

Die UR-Resolution ist korrekt, da sie immer das gleiche Ergebnis liefert wie eine entsprechende Folge von normalen Resolutionen. Sie ist nicht widerlegungsvollständig wie die folgende unerfüllbare Klauselmenge zeigt:

$$\{\{P, Q\}, \{P, \neg Q\}, \{\neg P, \neg Q\}, \{\neg P, \neg Q\}\}$$

Diese Klauselmenge enthält überhaupt keine unären Klauseln und kann damit nicht mit UR-Resolution widerlegt werden. Für eine eingeschränkte Klasse von Klauseln ist sie jedoch widerlegungsvollständig, und zwar für die sogenannten unär widerlegbare Klauselmengen. Diese Klauselmengen kann man widerlegen, indem man die Resolution dahingehend einschränkt, daß man fordert, daß eine der beteiligten Elternklauseln immer eine unäre Klausel ist, d.h. aus einem Literal besteht. Es gibt (bis jetzt) keine geeignete syntaktische Charakterisierung dieser Eigenschaft. Eine wichtige Unterklasse der unär widerlegbaren Klauselmengen sind Mengen von *Hornklauseln*, die in dieser Vorlesung noch behandelt werden.

Beispiel 7.1. UR-Resolution zur Aktionsplanung Typische “Blockswelt“:

Gesucht ist eine Aktionsfolge, die den initialen Zustand in den Zielzustand transformiert.

Axiomatisierung des Problems:
Es wird verwendet:

- $at(z, x, l)$: im Zustand z ist Block x auf Position l.
- $on(z, x, y)$: im Zustand z ist Block x auf Block y.
- $cl(z, x)$: im Zustand z ist Block x frei.
- $M(x, l)$: Block x geht nach Position l.
- $n(x, y)$: Zustand nach Aktion y vom Zustand x aus.
- Konstanten: S: Startzustand, A, B: Bausteine.

Allgemeine Regel:

$C1 : 1 \neq 2$

Ausgangszustand:

$C2 : at(S, A, 1)$

$C3 : at(S, B, 1)$

$C4 : cl(S, B)$

$C5 : on(S, B, A)$

Axiomatisierung der Aktion $M(x, l)$:

$C6 : \forall s, x, l : (cl(s, x) \Rightarrow at(n(s, M(x, l)), x, l))$

$C7 : \forall s, x, y, l_1, l_2 : (on(s, x, y) \wedge at(s, x, l_1) \wedge l_1 \neq l_2 \Rightarrow cl(n(s, M(x, l_2)), y))$

$C8 : \forall s, x, y, l(at(s, x, l) \Rightarrow at(n(s, M(y, l)), x, l))$

Ziel:

$$\exists z(at(z, A, 2) \wedge at(z, B, 2))$$

Klauselform:

$C1 : 1 \neq 2$

$C2 : at(S, A, 1)$

$C3 : at(S, B, 1)$

$C4 : cl(S, B)$

$C5 : on(S, B, A)$

$C6 : \neg cl(s, x), at(n(s, M(x, l)), x, l)$

$C7 : \neg on(s, x, y), \neg at(s, x, l1), l1 = l2, cl(n(s, M(x, l)), y)$

$C8 : \neg at(s, x, l), at(n(s, M(y, l)), x, l)$

$Z : \quad \neg at(z, A, 2), \neg at(z, B, 2)$

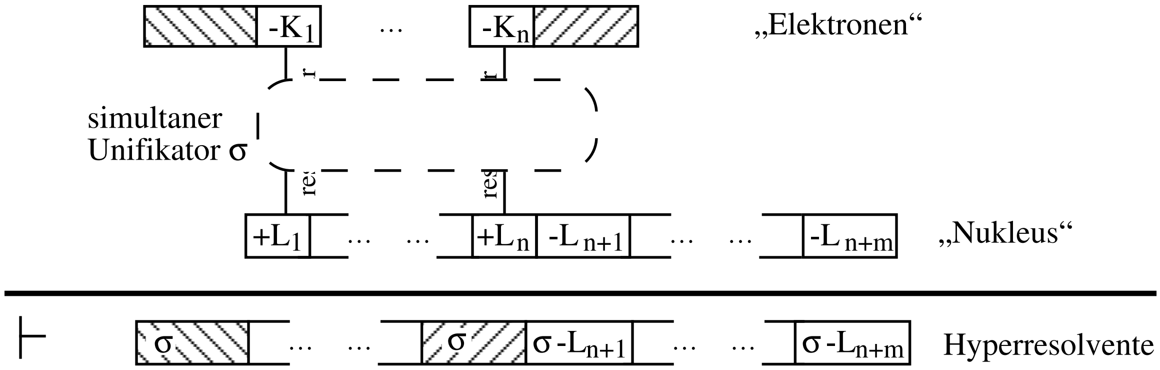
UR-Resolutionsfolge:

$$\begin{array}{lll}
C6 + C4 & \vdash R1 : at(n(S, M(B, l)), B, l) & \{s \mapsto S, x \mapsto B\} \\
C7 + C5, C2, C1 & \vdash R2 : cl(n(S, M(B, 2)), A) & \{s \mapsto S, x \mapsto B, y \mapsto A, l_1 \mapsto 1, l_2 \mapsto 2\} \\
C6 + R2 & \vdash R3 : at(n(n(S, M(B, 2)), M(A, l)), A, l) & \{s \mapsto n(S, M(B, 2)), x \mapsto B\} \\
C8 + R1 & \vdash R4 : at(n(n(S, M(B, l)), M(y, l)), B, l) & \{s \mapsto n(S, M(B, l)), x \mapsto B\} \\
Z + R3, R4 & \vdash R5 : \square & \{l \mapsto 2, z \mapsto n(n(S, M(B, 2)), M(A, 2))\}
\end{array}$$

Die gesuchte Aktionsfolge ist also $M(B, 2)M(A, 2)$.

7.3 Hyperresolution

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



Als "Nukleus" dient eine Klausel, die mindestens ein positives Literal enthält. Solche Klauseln gibt es in unerfüllbaren Klauselmengen immer. Für jedes positive Literal des Nukleus benötigt man ein sogenanntes "Elektron", eine Klausel mit nur negativen Literalen. Rein negative Klauseln kommen in unerfüllbaren Klauselmengen ebenfalls immer vor. Der Nukleus wird wieder simultan mit allen Elektronen resolviert, wodurch eine rein negative Klausel entsteht, die wiederum als Elektron im nächsten Hyperresolutionsschritt dienen kann. Die rein negativen Klauseln übernehmen hier also die gleiche Rolle wie die unären Klauseln in der UR-Resolution. Dual zu dieser sogenannten negativen Hyperresolution definiert man die positive Hyperresolution, bei der die Vorzeichen der Literale in Nukleus und Elektronen gerade vertauscht sind. Da im Normalfall eine negierte Behauptung nur negative Literale enthält und damit als Elektron für negative Hyperresolution verwendbar ist, eignet sich diese für Rückwärtsschließen von der Behauptung in Richtung Voraussetzungen, während positive Hyperresolution gerade umgekehrt von den Voraussetzungen in Richtung auf die Behauptung arbeiten kann. Beide Varianten der Hyperresolution (mit eingebauter Faktorisierung) sind widerlegungsvollständig für beliebige Klauselmengen.

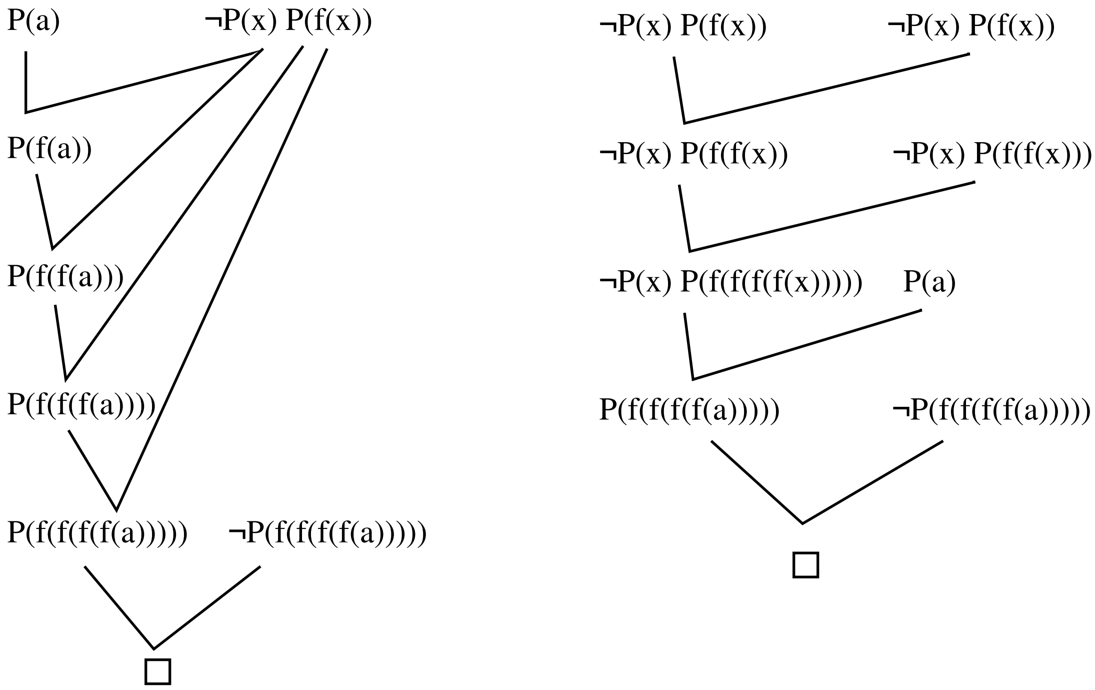
7.4 Input-Resolution und Unäre Resolution

unäre Resolution (englisch *unit resolution*) verbietet die Erzeugung von Resolventen zwischen zwei Elternklauseln, wenn beide mehr als ein Literal enthalten. Positiv formuliert muß also jede Resolvente mindestens eine unäre Elternklausel besitzen. Faktorisierung ist ebenfalls erlaubt. Diese Restriktion schränkt die möglichen Nachfolgezustände einer Klauselmenge stark ein, führt außerdem stets zu Resolventen mit weniger Literalen als sie die längere Elternklausel besitzt. Sie ist die Basisvariante zur UR-Resolution.

Beispiel 7.2. (Unäre Resolution)
Wir widerlegen die Klauselmenge $\{\{P(a)\}, \{\neg P(x), P(f(x))\}, \{\neg P(f(f(f(f(a)))))\}\}$, einmal mit unärer Resolution und dann ohne eine feste Strategie. Zur Darstellung benutzen wir einen sogenannten Widerlegungsbaum, bei dem jeder Knoten genau die beiden Elternklauseln als Vorgänger hat. Die Wurzel (diesmal wirklich unten) ist die leere Klausel.

Widerlegungsbäume

mit unärer Resolution mit allgemeiner Resolution



Wie man sieht, ist die unäre Widerlegung länger als die uneingeschränkte. Man kann das Beispiel leicht erweitern, so daß man sieht, daß die unäre Widerlegung exponentiell länger ist. Das bedeutet aber nicht, daß der Suchraum auch exponentiell größer ist, im Gegenteil: meist ist der Suchraum erheblich viel kleiner.

Die unäre Resolution ist im allgemeinen nicht widerlegungsvollständig, das heißt, die leere Klausel ist mit dieser Strategie nicht von allen unerfüllbaren Klauselmengen aus ableitbar. Immerhin ist die Widerlegungsvollständigkeit für eine wichtige Klasse von Klauselmengen gewährleistet, die die Klasse der Hornklauselmengen umfaßt und die man mangels einer syntaktischen Charakterisierung die *unär widerlegbare* Klasse nennt.

7.5 Eingabe-Resolution

Für die Klasse der unär widerlegbaren Klauselmengen ist auch eine andere wichtige Restriktionsstrategie widerlegungsvollständig, die *Eingaberesolution* (englisch *input resolution*). Diese verbietet die Erzeugung von Resolventen, deren

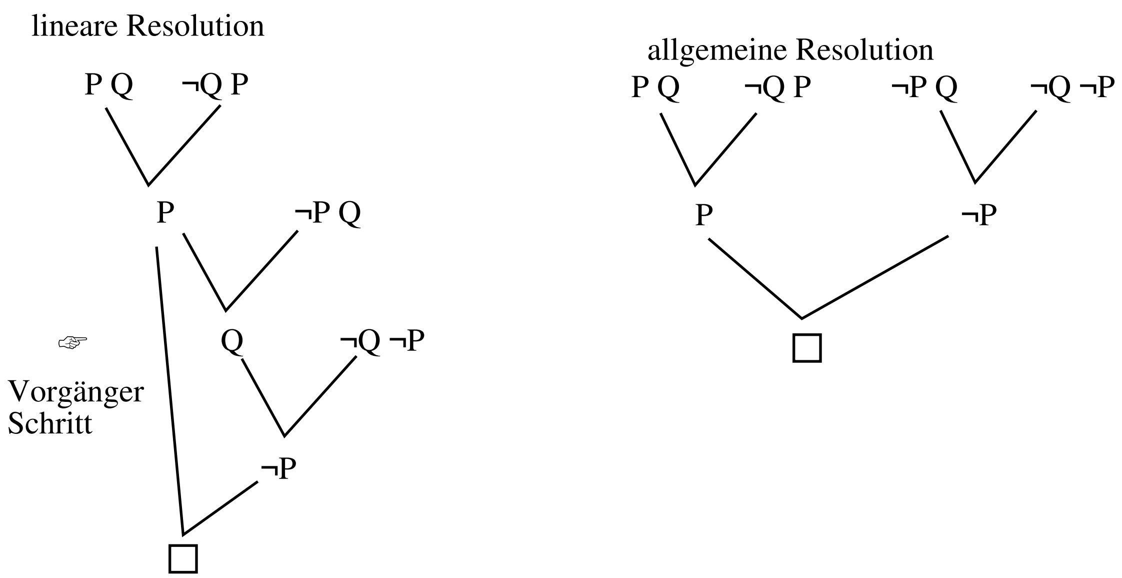
Elternklauseln beide Resolventen sind. Positiv formuliert muß also jede Resolvente wenigstens eine Elternklausel aus der initialen Klauselmenge besitzen. Der wesentliche Vorteil dieser Restriktion besteht daran, daß von allen möglichen Resolutionsschritten ein Resolutionsliteral a priori bekannt ist. Faktorisierung ist ebenfalls erlaubt. Insbesondere treten damit nur Unifikationen zwischen jeweils einem beliebigen und einem a priori bekannten Term auf, so daß man für jeden dieser bekannten Terme aus den initialen Klauseln einen speziellen Unifikationsalgorithmus "kompilieren" kann. Dieser ist in der Regel wesentlich effizienter als einer, der zwei beliebige Terme unifizieren können muß.

Es gilt: Eine Klauselmenge ist genau dann mit unärer Resolution widerlegbar, wenn sie mit Inputresolution widerlegbar ist. In diesem Sinne sind beide Strategien gleichwertig.

7.6 Lineare Resolution

Es gibt eine Reihe von Restriktionsstrategien, die die Grundidee der Eingaberesolution möglichst weitgehend erhalten wollen, aber widerlegungsvollständig für beliebige Klauselmengen sind. Die meisten basieren auf der *linearen Resolution.* Man arbeitet immer mit einer aktuellen Klausel, der Zentralklausel. Resolution mit Eingabe-Klauseln ist erlaubt, ebenso Faktorisierung, wobei diese auch implizit sein kann. Außerdem werden Resolutionsschritte zwischen zwei Resolventen in den Fällen zugelassen, in denen eine ein "Vorgänger" der anderen ist.

Beispiel 7.3. Lineare Resolution
Wir widerlegen die Klauselmenge $\{P, Q\}, \{\neg P, Q\}, \{P, \neg Q\}, \{\neg P, \neg Q\}$, einmal mit linearer Resolution und einmal mit allgemeiner Resolution:



Beispiel 7.4. Folgendes Beispiel zeigt, daß man Faktorisierung braucht. Die Klauselmenge sei $\{\{P(x), P(y)\}, \{\neg P(x), \neg P(y)\}\}$. Lineare Resolution ohne Faktorisierung ergibt immer eine Zentralklausel der Länge zwei, wobei auch keine automatische Verschmelzung auftritt.

Durch Verschärfungen dieser Bedingung sowie durch Beschränkungen für die als Resolutionsliterale zulässigen Literale einer Klausel ergeben sich verschiedene Varianten der linearen Resolution, zum Beispiel die sogenannte *SL-Resolution* .

7.7 SL-Resolution

SL-Resolution ist eine Restriktionsstrategie, die i.a. gekoppelt ist mit einer Ordnungsstrategie [?]. Die Bedeutung der SL-Resolution liegt darin, daß sie eine prozedurale "goal/subgoal" Sichtweise der Resolution erlaubt. SL-Resolution ist die Strategie, die der logischen Programmiersprache `PROLOG` zugrundeliegt. Im folgenden wird die SL-Resolution für Horn Klauseln behandelt. Im Falle der Hornklauselmengen ist die Faktorisierung nicht notwendig.

7.8 SL-Resolution für Horn Klauseln

Angenommen, wir haben eine Menge von initialen Horn Klauseln und eine "Zielklausel", d.h. ein negiertes Theorem ohne Kopfliteral. Beginnend mit der Zielklausel als "Zentralklausel" wählt SL-Resolution in jedem Schritt die letzte Resolvente als neue Zentralklausel und bestimmt die zulässigen Resolutionsschritte wie folgt: Wenn die aktuelle Zentralklausel $C = C_1 \vee C_2$ ist, wobei C_2 der Block von Literalen ist, der von der letzten "Seitenklausel" abstammt, wird aus C_2 ein Literal L ausgewählt und dafür eine Resolvente mit einer passenden Klausel aus der initialen Klauselmenge generiert. Daher resolviert SL-Resolution nur zwischen der letzten Resolvente und den Eingabeklauseln. Darüberhinaus werden diejenigen Literale zuerst wegresolviert, die als letzte in die Klausel hinein kamen. Die Auswahl des Literals in der Zentralklausel durch eine *Selektionsfunktion* ist deterministisch. Dem liegt die Erkenntnis zugrunde, daß die Reihenfolge, in der man Literale einer Klausel "wegresolviert" unerheblich ist – wenn es in einer Reihenfolge nicht geht, geht es auch in keiner anderen. Die Auswahl des Resolutionspartners ist dagegen nichtdeterministisch: eine erfolgreiche Resolutionskette ist ausreichend.

Beispiel 7.5. SL-Resolution mit Horn Klauseln

Prolog Notation:	Klausel Notation:
C1: $A(x,y) \Leftarrow P(x,y)$	$A(x,y) \vee \neg P(x,y)$
C2: $A(x,z) \Leftarrow P(x,y) \wedge A(y,z)$	$A(x,z) \vee \neg P(x,y) \vee \neg A(y,z)$
C3: $P(a,b)$	
C4: $P(b,c)$	
C5: $P(c,d)$	

Theorem: $\exists v : A(a,v) \wedge P(v,d)?$

Negiertes Theorem: $\neg A(a,v) \vee \neg P(v,d)$

In den beiden folgenden Bildern wird der Suchraum für dieses Beispiel und für zwei verschiedene Selektionsfunktionen dargestellt. Die Selektionsfunktionen sind:

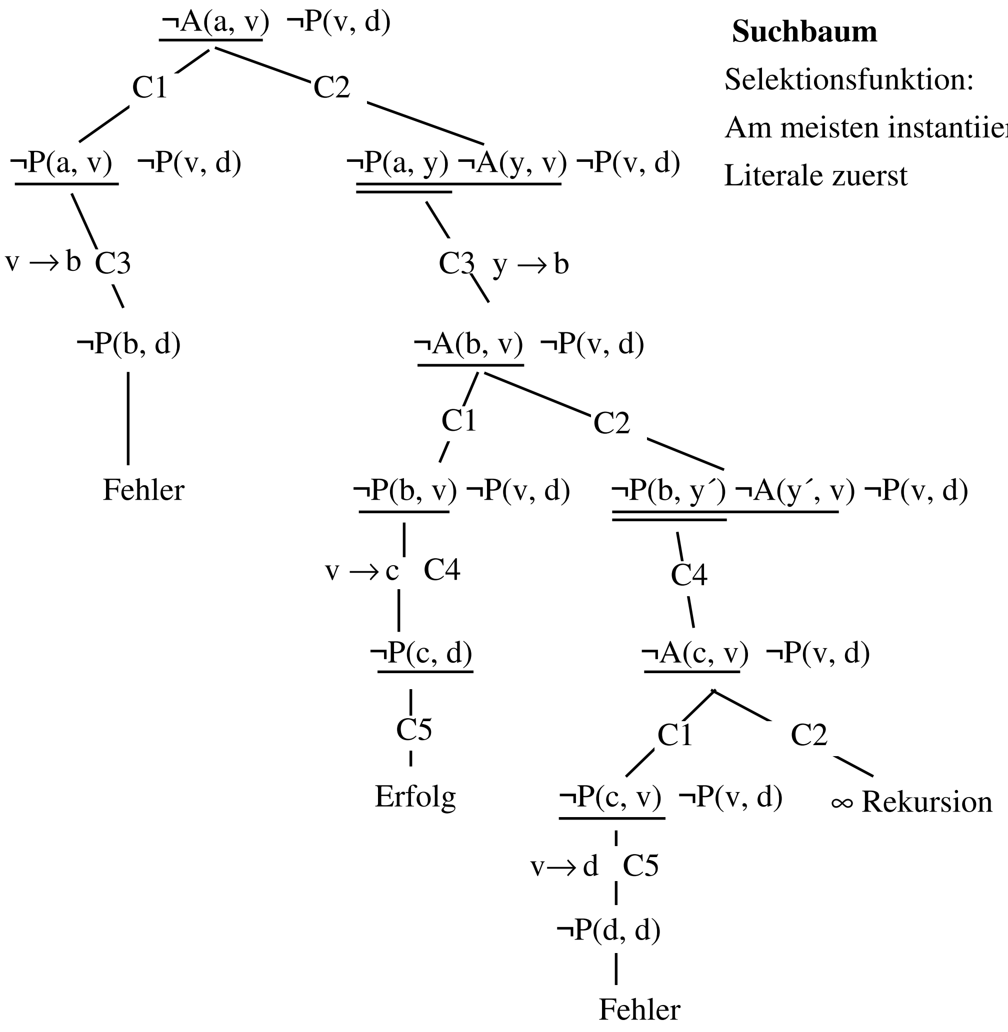
1. Das am stärksten instantiierte Literal zuerst

2. Das am wenigsten instantiierte Literal zuerst.

Die zuletzt eingeführten Literale, die zuerst wegresolviert werden, sind unterstrichen. Da SL-Resolution einen normalen Suchbaum generiert, können die normalen Suchalgorithmen angewendet werden [?]. Zum Beispiel Tiefensuche, wie sie in `Prolog` angewendet wird, hat den Vorteil, daß nur ein Ast zu einer Zeit entwickelt wird. Backtracking zu früheren Zuständen kann leicht durch Stacks implementiert werden. Tiefensuche ist jedoch nicht vollständig wenn der Suchraum unendliche Äste enthält. Bei Breitensuche und bei heuristischer Suche muß man dagegen alle offenen Knoten abspeichern, was sehr aufwendig sein kann.

Suchbaum

Selektionsfunktion:
Am meisten instantiierte
Literale zuerst



Suchbaum
Selektionsfunktion:
Am wenigsten instantiierte
Literale zuerst