Studentische Mitschrift von
Andre Egners
Last Update: 28. Juni 2006

<Logikprogrammierung>

<Sommersemester 2006>

<Prof. Dr. Giesl>

Zusammenfassung

Studentische Mitschrift zur Vorlesung Logikprogrammierung im Sommersemester 2006, gehalten von Prof. Dr. Giesl, des LuFGI-2 an der RWTH Aachen. Kommentare auf etwaige Fehler bitte an andre.egners@rwth-aachen.de senden.Die Mitschrift der Übung ist hier nicht zu finden, da die entsprechenden Lösugen vom Lehrstuhl veröffentlicht werden. Herzlichen Dank auch an Robert für deine Hilfe. Selbstverständlich erhebe ich keinerlei Anspruch auf Korrekheit der Mitschrift.

Inhaltsverzeichnis

4.4.2006

1 Einführung

Aufbau der Vorlesung:

1 Einführung in LP+Prolog

2 Syntax+Semantik der Prädikatenlogik: Wann folgt aus einer Formelmenge eine Formel?

3 Resolution (Technik um Folgerbarkeit automatisch zu untersuchen)

4 Logikprogrammierung

5 Prolog

6 Anwendung und Erweiterung der Logikprogrammierung

web: http://www-i2.informatik.rwth-aachen.de/lp06
Übungsschein: 50% der Punkte + Scheinklausur

Einsatzgebiet:

- Rapid Prototyping
- Deduktive Datenbanken
- Künstl. Intelligenz (vor allem Expertensysteme)

Bekannteste logische Programmiersprache: Prolog z.B. SWI-Prolog (freie Implemenierung) http://www.swi-prolog.org Darstellung der Wissensbasis durch Formel der Prädikatenlogik. (Prolog = Programming in Logic) Kommentare in Prolog: % oder /* .... */ Formel eines Logikprogramms: Klauseln

2 Arten von Klauseln:

- **Fakten** treffen Aussagen über Objekte (weiblich(monika)., verheiratet(werner, monika). weiblich, einstelliges Prädikatsymbolol, Name 1-stelliger Relation; verheiratet zweistelliges Prädikatsymbol; Prädikatsymbole und Objekte beginnen mit Kleinbuchstaben
- **Regeln** erlauben, aus bekannten Fakten neue Fakten zu schließen

Ausführung eines Logikkprogramms(LP): Stelle Anfragen. Prolog versucht Formeln zu Beweisen, indem es das Wissen aus der Wissensbasis verwendet. “Rechnen“ bedeutet in LP: “Beweisen“

```
?- maennlich(gerd).
Yes

?-verheiratet(gerd,monika).
No
```

“closed world assumption“: alles, was nicht aus der Wissensbasis folgt, ist falsch.
Um Programme ausführen zu können:

- Starte Prolog
- consult(datei). (datei.pl, enth. das Program) [datei:].
- Stelle Anfragen

Variablen

beginnen mit einem Großbuchstaben oder Unterstrich $(X, Y, Z, G, 172, ...)$ Variablen im Program stehen für alle möglichen Belegungen (sind allquantifiziert). Mensch(X). $\triangleq$ “alle sind Menschen“

```
?-mensch(gerd).
Yes

% folgt aus Wissensbasis, wenn {X/gerd}- subst verwendet wird

?-mensch(5).
Yes
```

Gleiche Variablen in der gleichen Klausel müssen gleich belegt werden: zusätzl.

Fakt:

```
mag(X,X).

?-mag(gerd,gerd).
Yes

?-mag(gerd,renate).
No
```

Fakt:

```
mag(X,Y).
```

Variablen in Anfrage: existenzquantifiziert

```
?-muttervon(X,susanne).
```

Gibt es Belegung von X, so das Aussage wahr ist.“Wer ist die Mutter von Susanne?“ Antwort nicht nur “Yes“, sondern Antwortsubstitution: X = renate

```
?-mutterVon(renate,Y).
% "'Welche Kinder hat renate?"'
Y=susanne     % <-- Prolog liefert erste gefundene Lï¿½ung
% "';"' eingeben um nach weiteren Lï¿½ungen zu suchen.
Y=peter
```

Klauseln im Programm werden von oben nach unten durchsucht.

```
?-muttzerVon(X,Y).
X=monika, Y=karin
```

Ein- und Ausgabe ist nicht durch das Programm festgelegt, sondern hängt von der Anfrage ab.

```
?-mensch(Y).
Y=Z  (Z ist neue Variable)
```

Prolog versucht möglichst allgemeine Lösungen zu finden. Lösungen müssen für alle Instanziierungen der verbleibenden Variablen wahr sein.

Kombination von Fragen:

Beispiel:

```
?-verheiratet(gerd,F), mutterVon(F, susanne).
```

“Gibt es eine Belegung von F, so dass sowohl verheiratet(gerd,F) als auch mutterVon(F, susanne) wahr ist?“ “Ist Gerd Vater von Susanne?“
Vorgehen von Prolog: Anfragen werden von links nach rechts bearbeitet.
Antwort auf Zwischenfrage: ?-verheiratet(gerd,mutterVon(F,susanne)), geht erstmal nicht da mutterVon kein Objekt zurückliefert /Antwort
Prolog bearbeitet erst “verheiratet(gerd,F)“ Dann bearbeitet Prolog “mutterVon(F,susanne)“
F=renate

```
?-mutterVon(Oma,Mama), mutterVon(Mama, aline).
```

Reihenfolge ist ungünstig → mehrfaches Rücksetzten (Backtracking) nötig, bis die richtige Lösung gefunden wird.
Regeln dienen dazu um aus bekannten Wissen neues Wissen herzuleiten. $p : -q, r$ heisst: p ist wahr falls q und r wahr sind; wenn q und r dann p

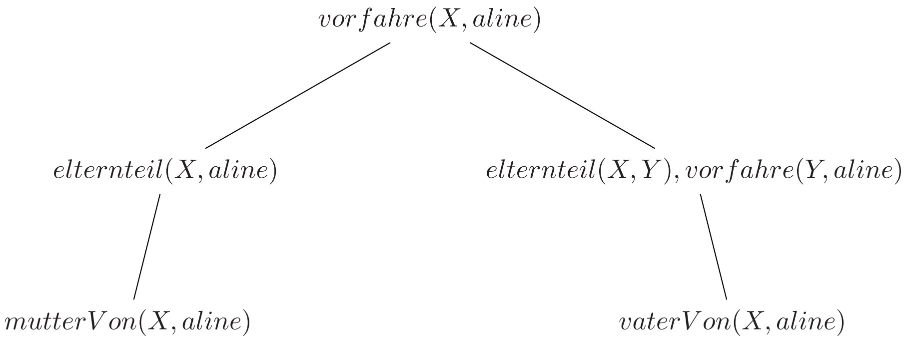
```
kopf der Regel  "falls"    Rumpf der Regel
vaterVon(V,K)     :-       verheiratet(V,F), mutterVon(F,K).
```

„?-vaterVon(gerd,Y).“ Baum entsteht, indem man Regelköpfe durch Regelrümpfe ersetzt. — Fakten entsprechen Regeln mit leerem Rumpf verheiratet(gerd,F), mutterVon(F,Y) — F/renate mutterVon(renate,Y) —Y/susanne Y/peter “Kasten“ ¡– leere Klausel, nicht mehr zu beweisen.
Y=susanne ¡- Belegung der Variablen aus der Anfrage.
Mehrere Regeln für ein Prädikat: $elternteil(X, Y)$ soll gelten, falls X Mutter oder Vater von Y ist.

```
elternteil(X,Y) :- mutterVon(X,Y).
elternteil(X,Y) :- vaterVon(X,Y).
```

⇒ realisiert Disjunktion im Programm.

Rekursive Regeln   vorfahre(V,X)



5.4.2006

2 Grundlagen der Prädikatenlogik

2.1 Syntax der Prädikatenlogik

Syntax legt fest, aus welchen Worten eine Sprache besteht.

Definitioninition 2.1.1 (Signatur) Eine Signatur (Σ, Δ) ist ein Paar mit $\Sigma = \bigcup_{n \in N} \Delta_n$ und $\Delta = \bigcup_{n \in N} \Delta_n$, wobei Σ_i, Δ_i paarweise disjunkt sind. Jedes $f \in \Sigma_n$ heißt n-stelliges Prädikatensymbol. Elemente von Σ_0 heißen auch Konstanten. Wir verlangen stehts $\Sigma_0 \neq \emptyset$.

Beispiel 2.1.2

(Σ, Δ) mit $\Sigma = \Sigma_0 \bigcup \Sigma_3$,
$\Delta = \Delta_1 \bigcup \Delta_2$ Signatur, die dem LP aus Kap.1 entspricht. $\Sigma_0 = \{monika, kalrin, ...\} \bigcup \mathbb{N}$,
$\Delta_1 = \{weiblich, maennlich, mensch\}$, $\Sigma_3 = \{datum\}$, $\Delta_2 = \{verheiratet, mutterVon, ...\}$
Datenobjekte werden in der Sprache der Prädikatenlogik als Terme repräsentiert.

Definition 2.1.3 (Term)

Sei (Σ, Δ) eine Signatur und $\mathcal{V}$ eine Menge von Variablen. Dann bezüglich $\tau(\Sigma, \mathcal{V})$ die Menge aller Terme über Σ und $\mathcal{V}$.
$\tau(\Sigma, \mathcal{V})$ ist die kleinste Menge mit:

- $\mathcal{V}$ Teilmenge von $\tau(\Sigma, \mathcal{V})$
- $f(t_1, .., t_n) \in \tau(\Sigma, \mathcal{V})$, falls $f \in \Sigma_n$ und $t_1, ..., t_n \in \tau(\Sigma, \mathcal{V})$ für ein $n \in \mathbb{N}$

$\tau(\Sigma)$ steht für $\tau(\Sigma, \emptyset)$ - Menge der Grundterme
$\mathcal{V}(t)$ ist Menge der Variablen in einem Term t.
Konvention: Funktions und Prädikatsymbole beginnen mit Kleinbuchstaben, Variablen mit Großbuchstaben.

Beispiel 2.1.4

Σ wie auf Folie, $\mathcal{V} = X, Y, Z, Oma, Mama, ...$
Beispiel für Terme: $X, monika, 5, datum(15, 10, 1966), datum(X, Oma, datum(15, 10, 1966))$
Aussagen werden in der Sprache der Prädikatenlogik als Formeln repräsentiert.

Definition 2.1.5 (Formel)

Sei (Σ, Δ) eine Signatur, $\mathcal{V}$ eine Menge von Variablen. Die Menge der atomaren Formeln über (Σ, Δ) und $\mathcal{V}$ sind definiert als $At(\Sigma, \Delta, \mathcal{V}) = p(t_1, ...t_n)$, $p \in \Delta_n$ für ein $n \in \mathbb{N}$, $t_1...t_n \in \tau(\Sigma, \mathcal{V})$.
$\mathcal{F}(\Sigma, \Delta, \mathcal{V})$ ist die Menge der Formeln. Sie ist die kleinste Menge mit:

- $At(\Sigma, \Delta, \mathcal{V}) \subseteq \tau(\Sigma, \Delta, \mathcal{V})$
- Wenn $\varphi \in \tau(\Sigma, \Delta, \mathcal{V})$, dann auch nicht $\varphi \in \tau(\Sigma, \Delta, \mathcal{V})$
- Wenn $\varphi_1, \varphi_2 \in \tau(\Sigma, \Delta, \mathcal{V})$, dann auch $\varphi_1 \wedge \varphi_2$; $\varphi_1 \vee \varphi_2$, $\varphi_1 \to \varphi_2 \in \tau(\Sigma, \Delta, \mathcal{V})$
- Wenn $X \in \mathcal{V}$ und $\varphi \in \tau(\Sigma, \Delta, \mathcal{V})$, dann $\forall X \varphi \exists X \varphi \in \tau(\Sigma, \Delta, \mathcal{V})$

$\mathcal{V}(\varphi)$ ist die Menge aller Variablen in φ. Eine Variable X ist frei in $\varphi \Leftrightarrow$

- φ ist atomare Formel und $X \in \mathcal{V}(\varphi)$
- $\varphi = \neg\varphi'$ und X ist frei in φ'

- $\varphi = \varphi_1 \circ \varphi_2$, $\circ \in \{\vee, \wedge, \rightarrow\}$ und X ist frei in φ_1 oder zu φ_2
- $\varphi = QY\varphi'$ mit $Q \in \{\forall, \exists\}$, $X \neq Y$ und X ist frei in φ'.

Eine Formel ist geschlossen, wenn sie keine freien Variblen enthält. Eine Formel ist quantorfrei, wenn sie weder "$\forall$" noch "$\exists$" enthält.

Beispiel 2.1.6

Formeln: `weiblich(monika)`$\in At(\Sigma, \Delta, \mathcal{V})$ `mutterVon(X,susanne)`$\in At(\Sigma, \Delta, \mathcal{V})$
`geboren(monia,datum((15.10.1966))`$\in At(\Sigma, \Delta, \mathcal{V})$
$\forall(verheiratet(gerd, F) \wedge mutterVon(F, K)) \in \tau(\Sigma, \Delta, \mathcal{V})$, freie Variable: K
`verheiratet(gerd,F)`und $\neg\forall F\ mutterVon(F, K) \in \mathcal{F}(\Sigma, \Delta, \mathcal{V})$ freie Variable:F, K
Schreibweise: $\forall X_1, \ldots, X_n\varphi$ steht für $\forall X_1(\forall X_2(\ldots(\forall_{X_n}\varphi)\ldots))$ Analog: $\exists X_1, \ldots, X_n\varphi$

Beispiel 2.1.7

Jedes Logikprogramm entspricht einer Formelmenge

- ":-" entspr. "$\leftarrow$"
- Variablen sind allquantifiziert
- "," entspr. "$\wedge$"

Definition 2.1.8 (Substitution)

Eine Abbildung $\sigma : \mathcal{V} \rightarrow \tau(\Sigma, \mathcal{V})$ heiï¿½ Substitution falls $\sigma(X) \neq X$ nur für endlich viele $X \in \mathcal{V}$ gilt. $DOM(\sigma) = \{X \in \mathcal{V} | \sigma(X) \neq X\}$ ist der Domain von sigma. Da $DOM(\sigma)$ endlich ist, ist eine Substitution als die endliche Menge $\{X(\sigma(X)|X \in DOM(\sigma)\}$ darstellbar. σ ist Grundsubtitution $\Leftrightarrow \mathcal{V}(\sigma(X)) = \emptyset$ für alle $X \in DOM(\sigma)$. σ ist eine Variablenumbenennung $\Leftrightarrow \sigma(X) \in \mathcal{V}\ \forall X \in \mathcal{V}$ und σ injektiv ist.
Beispiel: $\sigma = \{X/Y, Y/Z, Z/X\}$, $\sigma(X) = Y$, $\sigma(U) = U, \ldots$.
Substitutionen werden homomorph zu Abbildungen
$\sigma : \tau(\Sigma, \mathcal{V}) \rightarrow \tau(\Sigma, \mathcal{V})$ erweitert,
d.h.$\sigma(f(t_1, ..., t_n)) = f(\sigma(t_1), ..., \sigma(t_n))$.

Beispiel 2.1.9

$\sigma = \{X/datum(X, Y, Z), Y/monika, Z/datum(Z, Z, Z)\} \Rightarrow \sigma(datum(X, Y, Z)) = datum(datum(X, Y, Z), monika, datum(Z, Z, Z))$
Analog kann man Substitution auch auf Formeln anwenden:

- $\sigma(p(t_1, ...t_n)) = p(\sigma(t_1), ..., \sigma(t_n))$
- $\sigma(\neg\varphi) = \neg\sigma(\varphi)$
- $\sigma(\varphi_1 \circ \varphi_2) = \sigma(\varphi_1) \circ \sigma(\varphi_2)$ für $\circ \in \{\wedge, \vee, \rightarrow\}$
- $\sigma(QX\varphi) = QX\sigma(\varphi)$, für $Q \in \{\forall, \exists\}$, $\forall X mensch(X)$ und $\forall Y mensch(Y)$ sollten gleich behandelt werden falls $X \notin DOM(\sigma) \bigcup \mathcal{V}(\sigma(DOM(\sigma)))$
mit $\sigma(DOM(\sigma)) = \sigma(X)|X \in DOM(\sigma)$
- $\sigma(QX\varphi) = QX'\delta(\varphi)$ mit $\delta = X/X'$ sonst. Hierbei ist X' eine neue Variable
mit $X' \notin DOM(\sigma) \bigcup \mathcal{V}(\sigma(DOM(\sigma)))$

Beispiel 2.1.9 Fortsetzung

$\sigma(\forall Y verheiratet(X,Y)) = \forall Y' verheiratet(datum(X,Y,Z),Y')$
1.Problem: $Y \in DOM(\sigma)$
2.Problem: $Y \in \mathcal{V}(\sigma(X))$
Schreibweise: Statt $\sigma(datum(X,Y,Z))$:
$datum(X,Y,Z)[X/datum(X,Y,Z), Y/monika, Z/datum(Z,Z,Z)]$
$\sigma(t)$ ist Instanz von t und Grundinstanz, falls $\mathcal{V}(\sigma(t)) = \emptyset$

2.2 Semantik der Prädikatenlogik

Definitioninition 2.2.1 (Interpretation)

Für eine Signatur (Σ, Δ) ist eine Interpretation ein Tripel $I = (\mathcal{A}, \alpha, \beta)$ mit: $\mathcal{A} \neq \emptyset$ ist eine beliebige Menge ("Träger"). α ordnet jedem n-stelligen Funktionssymbol $f \in \Sigma_n$ eine Funktion α_f: $\mathcal{A} \times \ldots \times \mathcal{A} \to \mathcal{A}$ zu und jedem $p \in \Delta_n$ eine Menge $\alpha_p \subseteq \mathcal{A} \times \ldots \times \mathcal{A}$ (n-mal)
$\alpha_f \triangleq$ Deutung des Funktionssymbols f $\alpha_p \triangleq$ Deutung des Prädikatssymbols p $\beta \triangleq \mathcal{V} \to \mathcal{A}$ ist die Variablenbelegung der Interpretation I.
Zu jeder Interetation I erhält man eine Funktion I: $\tau(\Sigma, \mathcal{V}) \to \mathcal{A}$ mit:
$I(X) = \beta(X)\ \forall X \in \mathcal{V}\ I(f(t_1,...,t_n)) = \alpha_{f(I(t_1),...,I(t_n))}\ \forall f \in \Sigma_n,\ t_1,...,t_n \in \tau(\Sigma, \mathcal{V})$
$I(t)$ ist eine Interpretation des Terms t.
Für $X \in \mathcal{V}$ und $a \in \mathcal{A}$ ist $\beta[\![X/a]\!]$ die Variablenbelegung mit $\beta[\![X/a]\!](y) = \beta(y)\ \forall y \neq x \in \mathcal{V}$.
$I[\![X/a]\!]$ steht für $(\mathcal{A}, \alpha, \beta[\![X/a]\!])$ wenn $I = (\mathcal{A}, \alpha, \beta)$.
Eine Interpreation $I = (\mathcal{A}, \alpha, \beta)$ erfüllt eine Formel $\varphi \in \mathcal{F}(\Sigma, \Delta, \mathcal{V}) \Leftrightarrow$:

- $\varphi = p(t_1, \ldots, t_n)$ und $(I(t_1), \ldots, I(t_n)) \in \alpha_p$
- $\varphi = \varphi'$ und $I \not\models \varphi'$
- $\varphi_1 \circ \varphi_2$ und $(I \models \varphi_1) \circ (I \models \varphi_2)$ mit $\circ \in \{\vee, \wedge\}$
- $\varphi_1 \to \varphi_2$ und falls $I \models \varphi_1 \Rightarrow I \models \varphi_2$
- $\varphi = \forall X \varphi'$ und $I[\![X/a]\!] \models \varphi' \forall a \in \mathcal{A}$
- $\varphi = \exists X \varphi'$ und $I[\![X/a]\!] \models \varphi'$ für ein $a \in \mathcal{A}$

7.4.2006

Beispiel 2.2.2a

Betrachte (Σ, Δ) aus Beispiel 2.1.2 eine Interpretation $I = (\mathcal{A}, \alpha, \beta)$ für diese Signatur ist z.B.:
$\mathcal{A} = \mathbb{N}$
$\alpha_n = n, \forall n \in \mathbb{N}$
$\alpha_{monika} = 0, \alpha_{karin} = 1, \alpha_{renate} = 2, \ldots$
$\alpha_{mensch} = \mathbb{N}$
$\vdots$

- Eine Interpretation I ist ein Modell einer Formelmenge Φ $(I \models \Phi) \Leftrightarrow$ $I \models \varphi \forall \varphi \in \Phi$. Eine Interpretation I ist Modell von φ falls $I \models \varphi$.
- Zwei Formeln φ_1, φ_2 heißen äquivalent $\Leftrightarrow$ $I \models \varphi_1 \Leftrightarrow I \models \varphi_2$ für alle Interpretationen I.
- Eine Formel φ oder Formelmenge Φ ist erfüllbar, falls sie ein Modell hat.
- Eine Formel oder Formelmenge ist allgemeingültig, falls jede Interpretation von dieser Formel bzw. Formelmenge ist.

- Eine Interpretation ohne Variablenbelegung $S = (\mathcal{A}, \alpha)$ heißt Struktur, falls wir nur geschlossene Formeln betrachten.
- Begriffe gelten analog: $S \models \varphi$ mit $S = (\mathcal{A}, \alpha) \Leftrightarrow \exists$ Interpretation I, $I = (\mathcal{A}, \alpha, \beta) : I \models \varphi$.
- Genauso bei Grundtermen t.

Beispiel 2.2.2b

$\alpha_{datum}(\alpha_1, \beta(X), \alpha_{karin}) = 2$
$I \models$verheitratet(datum(1,X,karin),karin)?
Ist (2,1)$\in \alpha_{verheiratet}$? Ja $\Rightarrow$ verheiratet(...,...)$\in \alpha_{verheiratet}$

- Syntaktischer Begriff der Substitutionen:
  Abbildung von Variablen auf Terme: $\tau\mathcal{V} \to \mathcal{T}(\Sigma, \Delta)$
- Semantischer Begriff der Variablenbelegung:
  Abbildung von Variablen auf Trägerobjekte: $\beta : \mathcal{V} \to \mathcal{A}$

Lemma 2.2.3 (Substitutionslemma)

Sei $I = (\mathcal{A}, \alpha, \beta)$ eine Interpretation für (Σ, Δ).
Sei $\sigma = \{X_1/t_1, \ldots, X_n/t_n\}$ eine Substitution.
(a) $I(\sigma(t)) = I[\![X_1/I(t_1), \ldots, X_n/I(t_n)]\!](t)$ für alle Terme.
(b) $I \models \sigma(\varphi) \Leftrightarrow I[\![X_1/I(t_1), \ldots, X_n/I(t_n)]\!] \models \varphi$ für alle Formeln.
Beweis von (a):
Induktion über den Aufbau des Terms t:
IA: $t \in \mathcal{V}$ oder $t \in \Sigma_0$
IS: $t = f(t_1, \ldots, t_n)$, Hypothese: Aussage stimmt schon für $t_1, \ldots, t_n$.
Fall 1:
Falls $t = X_i$, dann $I(\sigma(X_i)) = I(t)$
$I[\![X_1/I(t_1), \ldots, X_n/I(t_n)]\!](X_i) = I(t_i)$.
Fall 2:
Falls $t = Y \in \{X_1, \ldots, X_n\}$, dann $I(\sigma(Y)) = I(Y)$
$I[\![X_1/I(t_1), \ldots, X_n/I(t_n)]\!](Y) = I(Y)$.

Beispiel 2.2.4

Sei I wie in Beispiel 2.2.2. $\sigma\{X/datum(1, X, karin)\}$.
$t = datum(X, Y, Z)$,
$I(\sigma(t)) = I(datum(datum(1, X, karin), Y, Z)) = I(2, 1, 2) = 5$.
$I[\![X/I(datum(1, X, karin))]\!](t) = I[\![X/2]\!](t)$
$= I[\![X/2]\!](datum(X, Y, Z)) = I[\![X/2]\!](datum(2, 1, 2)) = 5$

Definition 2.2.5 (Folgerbarkeit)

Aus einer gegebenen Formelmenge Φ folgt die Formel $\varphi \Leftrightarrow$ für alle Interpretationen mit $I \models \Phi$ und $I \models \varphi$ gilt. Falls Φ, φ keine freien Variablen enthalten, ist dies gleichbedeutend mit
$S \models \Phi \Leftrightarrow S \models \varphi$ für alle Strukturen S.
Statt $\emptyset \models \varphi$ schreibt man auch $\models \varphi$ (φ ist allgemeingültig).

Beispiel 2.2.6

Sei Φ die Formelmenge aus 2.1.7 (entspricht LP aus Kapitel 1).
Anfrage: ?-maennlich(gerd) bedeutet, dass man $\Phi \models$maennlich(gerd) beweisen muss.
?-mutterVon(X,susanne) bedeutet, dass man $\Phi \models \exists X$mutterVon(X,susanna) beweisen muss.

3 Resolution

Folgerbarkeit „$\Phi \models \varphi$“ ist semantisch definiert. Zur Untersuchung müsste man alle (unendliche vielen) Interpretationen betrachten.
Stattdessen: Führe Kalkül ein (z.B. Resolutionskalkül) der auf syntaktische Weise untersucht ob man aus Φ eine Formel φ herleiten kann.
$\Phi \models \varphi \rightsquigarrow$ Aus Φ ist φ herleitbar mit Kalkül.(vollständig)
Aus Φ ist φ herleitbar mit Kalkül $\rightsquigarrow \Phi \models \varphi$ (korrekt).

11.4.2006

Lemma 3.0.1 (Folgerbarkeit→Unerfüllbarkeit)

Seien $\varphi_1, \ldots, \varphi_n \in \mathcal{F}(\Sigma, \Delta, \mathcal{V})$. Dann gilt: $\{\varphi_1, \ldots, \varphi_n\} \models \varphi \Leftrightarrow \varphi_1 \wedge \ldots \wedge \varphi_n \wedge \neg\varphi$ unerfüllbar ist.
Beweis:
$\{\varphi_1, \ldots, \varphi_n\} \models \varphi$
$\Leftrightarrow$ für alle Interpretationen I mit $I \models \{\varphi_1, \ldots, \varphi_n\}$ gilt $I \models \varphi$
$\Leftrightarrow$ es gibt keine Interpretation I mit $I = \{\varphi_1, \ldots, \varphi_n\}$ und $I \models \neg\varphi$
$\Leftrightarrow \varphi_1 \wedge \ldots \wedge \varphi_n \wedge \neg\varphi$ ist unerfüllbar.

Beispiel 3.0.2

LP enthält Fakten. mutterVon(renate,susanne).
Anfrage: ?-muttervon(X,susanne)
z.z. {mutterVon(renate,susanne)} $\models \exists X$ mutterVon(X,susanne).
Statt dessen zeige Unerfüllbarkeit von: mutterVon(renate,susanne)$\wedge\neg\exists X$ mutterVon(X,susanne).

Nachweis der Folgerbarkeit/Unerfüllbarkeit ist unentscheidbar. Es existiert kein automatisches Verfahren, dass immer terminiert und Folgerbarkeit/Unentscheidbarkeit entscheidet.
Folgerbarkeit/Unentscheidbarkeit sind aber semi-entscheidbar. Es existiert ein automatisches Verfahren das terminiert und Folgerbarkeit/Unentscheidbarkeit nachweist, falls Folgerbarkeit/Unentscheidbarkeit gilt. Aber wenn Folgerbarkeit/Unentscheidbarkeit nicht gilt, dann terminiert das Verfahren eventuell nicht.
Durch Automatisierung des Resolutionskalküls könnte man z.B. solch ein Semi-Entscheidungsverfahren erhalten.
Einführung des Resolutionskalküls in 5 Schritten.

1. Überführe Formel in Skolem-Normalform (SNF):
   $\forall X_1, \ldots, X_n \varphi$ (φ quantorfrei, $\mathcal{V}(\varphi) \subseteq \{X_1, \ldots, X_n\}$
2. Zeige, dass man sich bei Formeln in SNF auf Herbrand-Interpretationen einschränken kann.
   $\Rightarrow$ Erstes (ineffizientes) Semi-Entscheidungsverfahren.
3. Aussagenlogische Resolution $\Rightarrow$ Zweites Verfahren
4. Prädikatenlogische Resolution $\Rightarrow$ Drittes Verfahren
5. Einschränkung der Resolution $\Rightarrow$ Viertes Verfahren

3.1 Skolem-Normalform

Zwei Schritte:
(1) Übersetzung von Formeln in Pränex-Normalform
(2) Weitere Überführung in SNF

Definition 3.1.1. (Pränex-Normalform)

Eine Formel φ ist in PNF $\Leftrightarrow$ sie die Gestalt $Q_1X_1 \dots Q_nX_n\psi$ mit $Q_i \in \{\exists, \forall\}$ und ψ quantorfrei.

Satz 3.1.2 (Überführung in PNF)

Zu jeder Formel ψ läßt sich automatisch eine äquivalente Formel ψ' in PNF finden.
Beweis:
(1) Ersetze zuerst alle TF $\varphi_1 \rightarrow \varphi_2$ durch $\neg\varphi_1 \vee \varphi_2$. Danach wende Algorithmus PRAENEX an.
Alg. PRAENEX
Eingabe: Formel φ ohne „$\rightarrow$“
Ausgabe: Zu φ äquivalente Formel in PNF.

- Falls φ quantorfrei ist, dann liefere φ zurück.
- Falls $\varphi = \neg\varphi_1$, so berechne PRAENEX(φ_1) $= Q_1X_1 \dots Q_nX_n\varphi$
  Liefere $\neg Q_1X_1 \dots Q_nX_n\neg\varphi$ zurück.
- Falls $\varphi = \varphi_1 \circ \varphi_2$, $\circ \in \{\wedge, \vee\}$ so
  berechne PRAENEX(φ_1)$=Q_1X_1 \dots Q_nX_n\psi_1$ und PRAENEX(φ_2) $= R_1Y_1 \dots R_nY_n$. Durch Umbenennung gebundener Variablen erreichen wir, dass $X_1 \dots X_n$ nicht in $R_1Y_1 \dots R_nY_n$ auftreten und das $Y1 \dots Y_n$ nicht in $Q_1X_1 \dots Q_nX_n$ auftreten.
  Liefere $Q_1X_1 \dots Q_nX_nR_1Y_1 \dots R_nY_n\psi_1 \circ \psi_2$ zurück.
- Falls $\varphi = QX\varphi_1$ mit $Q \in \{\exists, \forall\}$, so berechne PRAENEX($\varphi_1$) $= Q_1X_1 \dots Q_nX_n\psi_1$
  Durch umbenennen gebundener Variablen erreichen wir, dass $X_1, \dots X_n$ verschieden von X sind. Liefere $Q_1X_1 \dots Q_nX_n$ zurück.

Beispiel 3.1.3

$\neg\exists X(verheiratet(X,Y) \vee \neg\exists Y mutterVon(X,Y))$
$\Rightarrow verheiratet(X,Y) \vee \forall Y \neg mutteVon(X,Y)$
$\Rightarrow verheiratet(X,Y) \vee \forall Z \neg mutterVon(X,Z)$
$\Rightarrow \forall Z(verheiratet(X,Y) \vee \neg mutterVon(X,Z)$
$\Rightarrow \forall X \forall Z(verheiratet(X,Y) \vee mutterVon(X,Z))$
$\Rightarrow \forall X \exists Z \neg(verheiratet(X,Y) \vee \neg mutterVon(X,Z))$

Beispiel 3.1.4 (Fortsetzung Beispiel 3.0.2)

mutterVon(renate,susanne)$\wedge\neg\exists X$mutterVon(X,susanne).
$\Rightarrow \forall X$mutterVon(renate,susanne)$\wedge\neg$mutterVon(X,susanne).

Definition 3.1.5 (Skolem-Normalform)

Eine Formel φ ist in SNF $\Leftrightarrow$ φ ist geschlossen und hat die Gestalt $\forall X_1 \dots X_n\psi$ mit ψ quantorfrei.
Existiert zu jeder Formel eine äquivalente Formel in SNF? *Nein!* weiblich(X)$\Rightarrow \exists X$weiblich(X).
Es existiert zu jeder Formel φ eine erfüllbarkeitsäquivalente Formel in SNF.

Satz 3.1.6 (Überführung in SNF)

Zu jeder Formel φ läßt sich automatisch eine Formel φ' in SNF konstruieren, so dass φ erfüllbar$\Leftrightarrow$ φ' erfüllbar.
Beweis:
Sei φ in PNF. Seien $X_1, \dots, X_n$ die freien Variablen von φ_1. Dann wird φ_1 weiter überführt in $\varphi_2 : \exists X_1 \dots X_n\varphi_1$

φ_1 und φ_2 sind erfüllbarkeitsäquivalent. Danach werden die Existenzquantoren Schrittweise, von aussen nach innen beseitigt. Falls $\varphi_2 = \forall X_1 \dots X_n \exists Y \psi$ ($n \geq 0, \psi$ in PNF) dann ersetze φ_2 durch φ_2':
$\forall X_1 \dots X_n \psi[Y/f(X_1, \dots, X_n)]$ wobei f ein neues n-stelliges Funktionssymbol ist.
Es gilt: φ_2 und φ_2' sind erfüllbarkeitsäquivalent.
Durch mehrfache Wiederholung dieser Technik werden alle $\exists$ beseitigt.

21.4.2006

Beispiel 3.1.7 (Fortsetzung von Beispiel 3.1.3)

PNF: $\forall X \exists Z \neg(ver(X,Y) \vee \neg mV(X,Z))$
$\Rightarrow \exists Y \forall X \exists Z \neg(ver(X,Y) \vee \neg mV(X,Z))$
$\Rightarrow \forall X \exists Z \neg(ver(X,a) \vee \neg mV(X,Z))$
$\Rightarrow \forall X \neg(ver(X,a) \vee \neg mV(X,b(X)))$

3.2 Herbrand-Strukturen

Bisher: Um Unerfüllbarkeit einer geschlossenen Formel zu zeigen, muss man alle (unendliche viele) Strukturen durchtesten.
3 Freihheitsgrade:

- Träger $\mathcal{A}$
- Deutung der Funktionssymbole α_f, $\forall f \in \Sigma$
- Deutung der Prädikatssymbole α_p, $\forall p \in \Delta$

Wir zeigen: Bei Formeln in SNF kann man die ersten beiden Freiheitsgrade festlegen.
$\Rightarrow$ Suchraum nach einer Struktur, die die Formel erfüllt, wird wesentlich kleiner.
$\Rightarrow$ Einschränkung auf Herbrand-Strukturen.

Definition 3.2.1 (Herbrand-Struktur)

Sei (Σ, Δ) eine Signatur, dann hat eine Herbrand-Struktur (Σ, Δ) die Gestalt:
$(\mathcal{T}(\Sigma, \alpha)$, wobei $\forall f \in \Sigma_n$ mit $n \in \mathbb{N}$ gilt:
$\alpha_f(t_1, \dots, t_n) = f(t_1, \dots, t_n)$.
Falls die Herbrand-Struktur Modell einer Formel φ ist, dann bezeichnet man sie als Herbrand-Modell von φ.
$S(t) = t$, $\forall t \in \mathcal{T}(\Sigma)$: *Grundterme werden als sich selbst gedeutet*
Deutung der Prädikatssymbole kann frei gewählt werden.

Beispiel 3.2.2 (Signatur von Folie „Signatur des Logikprogramms")

$S = (\mathcal{T}(\Sigma), \alpha)$ für diese Signatur.
$\alpha_n = n$, $\forall n \in \mathbb{N}$.
$\alpha_{monika} = monika$
...
$\alpha_{datum}(t_1, t_2, t_3) = datum(t_1, t_2, t_3)$
$\alpha_{weiblich} = \{monika, karin, \dots\}$
Zur Untersuchung der Unerfüllbarkeit kann man sich auf Herbrand-Strukturen beschränken.

Satz 3.2.3 (Erfüllbarkeitstest durch Herbrand-Sturkturen)

Sei $\Phi \subseteq\in \mathcal{F}(\Sigma, \Delta, \mathcal{V})$ eine Menge von Formeln in SNF. Dann ist Φ erfüllbar $\Leftrightarrow$ Φ hat ein Herbrand-Modell.
Beweis:
„$\Leftarrow$“: Trivial.
„$\Rightarrow$“: Sei $S' = (\mathcal{A}, \alpha')$ ein Modell von Φ. Wir zeigen, dass die Herbrand-Struktur dann auch Modell von Φ ist. Für alle $f \in \Sigma_n$ gilt $\alpha'_f(t_1, \ldots, t_n)$.
Für alle $p \in \Delta_n$ definieren wir α' mit:
$(t_1, \ldots, t_n) \in \alpha'_p \Leftrightarrow (S(t_1), \ldots, S(t_n)) \in \alpha_p$
Wir zeigen, dass für jede Formel φ in SNF gilt:
$S \models \varphi \Rightarrow S' \models \varphi$
Induktion über die Anzahl n, der allquantifizierten Variablen.
IA: n=0; φ ist quantor- und variablenfrei. Hier gilt sogar $S \models \varphi \Leftrightarrow S' \models \varphi$.
IS: n>0: $\forall X_1, \ldots X_{n-1}\psi$ enthält evtl. die freie Variable X_n und ist nicht in SNF.
$S \models \forall X_1, \ldots, X_n\psi \Leftrightarrow S[\![X_n/a]\!] \models \forall X_1, \ldots, X_{n-1}\psi$, $\forall a \in \mathcal{A}$
$\Rightarrow S[\![X_n/S(t)]\!] \models \forall X_1, \ldots, X_{n-1}\psi$, $\forall t \in \mathcal{T}(\Sigma)$
$\Leftrightarrow S \models \forall X_1, \ldots, X_{n-1}\psi[X_n/t]$, $\forall t \in \mathcal{T}(\Sigma)$
$\Rightarrow S' \models \forall X_1, \ldots, X_{n-1}\psi[X_n/t]$, $\forall t \in \mathcal{T}(\Sigma)$
$\Leftrightarrow S'[\![X_n/S(t)]\!] \models \forall X_1, \ldots, X_{n-1}\psi$, $\forall t \in \mathcal{T}(\Sigma)$
$\Leftrightarrow S'[\![X_n/t]\!] \models \forall X_1, \ldots, X_{n-1}\psi$, $\forall t \in \mathcal{T}(\Sigma)$
$\Leftrightarrow S' \models \forall X_1, \ldots, X_n\psi$

Beispiel 3.2.4 (Satz 3.2.3 gilt nur für Formeln in SNF)

Gegenbeispiel: (Σ, Δ) mit $\Sigma = \Sigma_0 = a$ und $\Delta = \Delta_1 = p$
Formelmenge $\{p(a), \exists X \neg p(X)\}$ ist erfüllbar durch die Struktur$(\{0, 1\}, \alpha)$ mit $\alpha_a = 0$ und $\alpha_p = \{0\}$.
Grund: Träger enthält ein Objekt „1“ das nicht als Deutung von Grundtermen erreicht werden kann. Herbrand-Struktur$(\{a\}, \alpha')$ mit $\alpha'_a = a$ und $\alpha'_p = a$ oder $\alpha'_p = \emptyset$

Reduziere das Unerfüllbarkeitsproblem weiter auf Unerfüllbarkeitsuntersuchung einer (unendlichen) Menge ohne Variablen. Dies entspricht der Überprüfung der Unerfüllbarkeit für eine (unendliche) Menge aussagenlogischer Formeln. Da Herbrand-Strukturen zur Untersuchung der Unerfüllbarkeit ausreichen, ersetze allquantifizierte Variablen durch alle möglichen Grundterme!

Definition 3.2.5 (Herbrand-Expansion einer Formel)

Sei φ eine Formel in SNF, mit $(\varphi \in \mathcal{F}(\Sigma, \Delta, \mathcal{V}))$, $\varphi = \forall X_1, \ldots, X_n\psi$ mit ψ quantorfrei.
Die Formelmenge $E(\varphi)$ heißt Herbrand-Expansion von φ:
$E(\varphi) = \{\psi[X_1/t_1, \ldots, X_n/t_n] \mid t_1, \ldots, t_n \in \mathcal{T}(\Sigma)\}$ (Menge aller Grundinstanzen von von ψ)

Beispiel 3.2.6

$\varphi = \forall X$(mutterVon(renate,susanne)$\wedge\neg$mutterVon(X,susanne))
$E(\varphi) = \{$mutterVon(renate,susanne)$\wedge\neg$mutterVon(karin,susanne),
mutterVon(renate,susanne)$\wedge\neg$mutterVon(renate,susanne),$\ldots\}$
$E(\varphi)$ ist offensichtlich unerfüllbar!

Satz 3.2.7

Sei φ in SNF. Dann gilt φ ist erfüllbar $\Leftrightarrow$ Herbrand-Expansion von φ ist erfüllbar.
Beweis: $\forall X_1, \ldots, X_n\psi$ erfüllbar
$\Leftrightarrow S \models \forall X_1, \ldots, X_n\psi$ für eine Herbrand-Struktur S *(Satz 3.2.3)*

$\Leftrightarrow S[\![X_1/t_1,\ldots,X_n/t_n]\!] \models \psi, \forall t_1,\ldots,t_n \in \mathcal{T}(\Sigma)$, *($t_1 = S(t_1)$ usw.)*
$\Leftrightarrow S \models \psi[X_1/t_1,\ldots,X_n/t_n], \forall t_1,\ldots,t_n \in \mathcal{T}(\Sigma)$, *(Lemma 2.2.3)*
$\Leftrightarrow S \models E(\varphi)$
$\Leftrightarrow E(\varphi)$ ist erfüllbar *(Satz 3.2.3)*

Prädikatenlogische Formeln ohne Variablen entsprechen aussagenlogischen Formeln. Man kann jede atomare Teilformel $p(t_1,\ldots,t_n)$ als aussagenlogische Variable ansehen, die wahr oder falsch sein kann.
$\rightarrow$ Beispiel 3.2.6
Statt mutterVon(renate,susanne): $V_{mutterVon(renate,susanne)}$ und
statt mutterVon(karin,susanne): $V_{mutterVon(karin,susanne)}$
Diese Variablen können nun entwerder als wahr, oder als falsch interpretiert werden.

Zur Unerfüllbarkeit von $E(\varphi)$: Zeige die Unerfüllbarkeit der folgenden aussagenlogischen Formelmenge:
$\{V_{mutterVon(renate,susanne)} \wedge \neg V_{mutterVon(karin,susanne)}, V_{mutterVon(renate,susanne)} \wedge \neg V_{mutterVon(renate,susanne)}\}$

Algorithmus von Gilmore

Ziel: Überprüfe $\{\varphi_1,\ldots,\varphi_k\} \models \varphi$ (*)

1. Setze $\psi = \varphi_1 \wedge \ldots \wedge \varphi_k \wedge \neg\varphi$ ((*)$\Leftrightarrow \psi$ unerfüllbar)
2. Überführe ψ in SNF. *(Satz 3.1.2 + 3.1.6)*
3. Wähle Aufzählung $\{\psi_1,\psi_2,\ldots\} = E(\psi)$ *(Satz 3.2.7)*
4. Prüfe ob ψ_1, $\psi_1 \wedge \psi_2, \psi_1 \wedge \psi_2 \wedge \psi_3, \ldots$ aussagenlogisch erfüllbar ist. Falls eine nicht erfüllbar ist ENDE und gib „Yes“ zurück.

Dies ist ein Semi-Entscheidungsverfahren (jede unenliche unerfüllbare AL-Formelmenge hat eine endliche unerfüllbare Teilmenge, siehe KP-Satz der AL). Nachteil ist, dass die Ersetzung von Variablen durch Grundterme nicht zielgerichtet ist und somit ineffizient ist. Zudem terminiert der Algorithmus nicht falls $\{\varphi_1,\ldots,\varphi_k\} \not\models \varphi$.

25.4.2006

3.3 Grundresolution

Um $\forall X_1,\ldots,X_n\psi$ in SNF mit Resolutionkalül auf Unerfüllbarkeit zu untersuchen, muss ψ zunächst in KNF überführt werden.$\Rightarrow$ Darstellung als Klauselmenge.

Definition 3.3.1 (KNF, Literal, Klausel)

Eine Formel ψ ist in KNF $\Leftrightarrow \psi$ ist quantorfrei und hat folgende Gestalt:
$(L_{1,1} \vee \ldots \vee L_{1,n_1}) \wedge \ldots \wedge (L_{m,1} \vee \ldots \vee L_{m,n_m})$. $L_{i,j}$ sind Literale, d.h. atomare, oder negierte atomare Formeln (d.h. $p(t_1,\ldots,t_n)$ oder $\neg p(t_1,\ldots,t_n)$). Zu einem Literal L definieren wir ein Negat $\bar{L}$ als:
$\bar{L} = \neg A$, falls $L = A \in At(\Sigma,\Delta,\mathcal{V})$ oder
$\bar{L} =$ A, falls $L = \neg A \in At(\Sigma,\Delta,\mathcal{V})$

Eine Menge von Literalen heißt Klausel. Jede Formel ψ in KNF wie oben, entspricht der zugehörigen Klauselmenge $\mathcal{K}(\psi)$:
$\mathcal{K}(\psi) = \{\{L_{1,1},\ldots,L_{1,n_1}\},\ldots,\{L_{m,1},\ldots,L_{m,n_m}\}\}$
Eine Klausel steht für eine allquantifizierte Disjunktion ihrer Literale
$(\{L_{1,1},\ldots,L_{1,n_1}\} \triangleq \forall\ldots(L_{1,1} \vee \ldots \vee L_{1,n_1}))$.

Eine Klauselmenge steht für die Konjunktion ihrer Klauseln
$\Rightarrow$ „Folgerbarkeit", „Erfüllbarkeit" ist auch für Klauselmengen definiert.
Wir betrachten meist nur endliche Klauselmengen. Die leere Klausel schreiben wir als „$\Box$".
$\Box$ ist nach Definition unerfüllbar (leere Disjunktion).

Satz 3.3.2 (Überführung in KNF)

Zu jeder quantorfreien Formel ψ existiert eine äquivalente Formel ψ' in KNF, die man automatisch konstruieren kann.
Beweis:
Ersetze zunächst alle $\psi_1 \to \psi_2$ durch $\neg\psi_1 \vee \psi_2$. Überführung der verbleibenden ψ durch Algorithmus KNF:

- Falls ψ atomar, gibt ψ zurück
- Falls $\psi = \psi_1 \wedge \psi_2$, dann gibt KNF($\psi_1$)$\wedge$KNF($\psi_2$) zurück
- Falls $\psi = \psi_1 \vee \psi_2$, dann berechne KNF$(\psi_1) = \bigwedge_{i\in\{1,\ldots,m_1\}} \psi_i'$ und KNF$(\psi_1) = \bigwedge_{j\in\{1,\ldots,m_2\}} \psi_j''$.
  Gib $\bigwedge_{\substack{i\in\{1,\ldots,m_1\}\\ j\in\{1,\ldots,m_2\}}} \psi_i' \vee \psi_j''$ zurück.
- Falls $\psi = \neg\psi_1$, dann berechne KNF$(\psi_1) = \bigwedge_{i\in\{1,\ldots,m\}}(\bigvee_{j\in\{1,\ldots,n_i\}} L_{i,j})$
  De-Morgan liefert $\bigvee_{i\in\{1,\ldots,n\}}(\bigwedge_{j\in\{1,\ldots,n_i\}} \bar{L}_{i,j})$
  Gib $\bigwedge_{\substack{j_1\in\{1,\ldots,n_1\}\\ \vdots \\ j_m\in\{1,\ldots,n_m\}}} \bar{L}_{1,j_1} \vee \ldots \bar{L}_{m,j_m}$

Beispiel 3.3.3

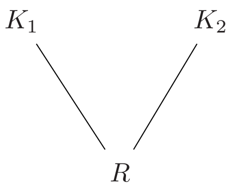
Sei Δ_0=$\{p,q,r\}$, Formel: $\neg(\neg p \wedge (\neg q \vee r))$
$\Rightarrow (p \vee q) \wedge (p \vee \neg r) := \psi$ (KNF)
$\Rightarrow \mathcal{K}(\psi) = \{\{p,q\},\{p,\neg r\}\}$
Ziel: Nachweis von Unerfüllbarkeit einer Klauselmenge. (Effizienter als Gilmore-Algorithmus)

Definition 3.3.4 (Aussagenlogische Resolution)

Seien K_1, K_2 variablenfreie Klauseln. Dann ist der Resolvent R von $K_1, K_2 \Leftrightarrow$ es gibt ein $L \in K_1$ mit $\bar{L} \in K_2$ und $R = (K_1 \setminus L \cup K_2 \setminus \bar{L})$. Für eine Klauselmenge $\mathcal{K}$ definieren wir:
$Res(\mathcal{K}) = \mathcal{K} \cup \{R \mid R$ ist Resolvent zweier Klauseln aus $\mathcal{K}\}$
$Res^0(\mathcal{K}) = \mathcal{K}$
$Res^{n+1}(\mathcal{K}) = Res(Res^n(\mathcal{K}))$
$Res^*(\mathcal{K}) = \bigcup_{n\geq 0} Res^n(\mathcal{K})$ ist die Menge aller Klauseln durch Resolution aus $\mathcal{K}$ hergeleiteten Klauseln.
Ziel: $\Box$ Klausel durch Resolution aus $\mathcal{K}$ herleiten. (Dann ist $\mathcal{K}$ unerfüllbar)
Es gilt $\Box \in Res^*(\mathcal{K}) \Leftrightarrow$ es gibt eine Folge von Klauseln $K_1,\ldots,K_m$ mit $K_m = \Box$ und $\forall i 1 \leq i \leq m$:

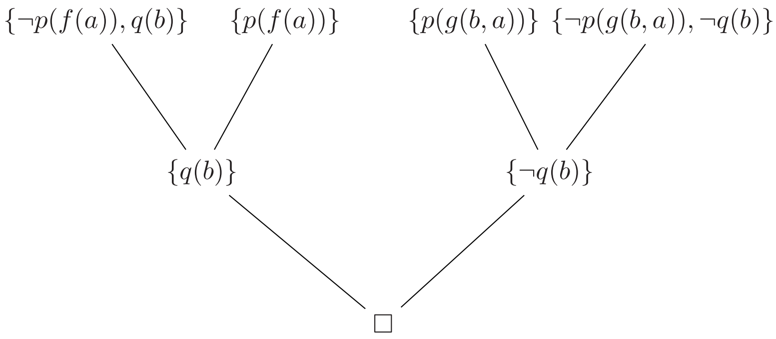
- $K_i \in \mathcal{K}$ oder
- K_i ist Resolvent von $K_j, K_k,\ j,k \leq i$

Darstellung von Resolutionsbeweisen:
Resolvent R entsteht durch Resolution von K_1 und K_2



Beispiel 3.3.5

$\Delta_1 = \{p, q\}$, $\Sigma_0 = \{a, b\}$, $\Sigma_1 = \{f\}$, $\Sigma_2 = \{g\}$



Zeige, dass der Resolutionskalkül korrekt und vollständig ist.
$\mathcal{K}$ unerfüllbar$\Leftrightarrow \square \in Res^*(\mathcal{K})$

Lemma 3.3.6 (Aussagenlogisches Resolutionslemma)

Sei $\mathcal{K}$ eine variablenfreie Klauselmenge, $K_1, K_2 \in \mathcal{K}$. Falls R Resolvent von K_1 und K_2 ist, dann sind $\mathcal{K}$ und $\mathcal{K} \cup \{R\}$ äquivalent.
Beweis:

- $\mathcal{K} \cup \{R\} \models \mathcal{K}$: Jede Struktur die $\mathcal{K} \cup \{R\}$ erfüllt, erfüllt offensichtlich $\mathcal{K}$ (Klauselmenge entspricht Konjunktion ihrer Klauseln)

- $\mathcal{K} \models \mathcal{K} \cup \{R\}$: Sei S Struktur mit $S \models \mathcal{K}$. Annahme: $S \not\models R$. Es existiert $L \in K_1$, $\bar{L} \in K_2$, $R = (K_1 \setminus L) \cup (K_2 \setminus \bar{L})$

  1. $S \models L$:
     Da $S \models L$, gilt $S \models K_2$. Da $S \not\models \bar{L}$ gilt $S \models K_2 \setminus \bar{L}$. Daher auch $S \models R$. Widerspruch!
  2. $S \not\models L$, d.h. $S \models \bar{L}$. Da $S \models \mathcal{K}$, gilt $S \models K_1 \Rightarrow S \models K_1 \setminus L \Rightarrow S \models R$. Widerspruch!

28.4.2006

Satz 3.3.7 (Korrektheit und Vollständigkeit der aussagenlogischen Resolution)

Sei $\mathcal{K}$ eine Variablenfreie Klauselmenge. Dann gilt $\mathcal{K}$ unerfüllbar $\Leftrightarrow \Box \in Res^*(\mathcal{K})$
<u>Beweis:</u>
" $\Leftarrow$ " (Korrektheit)
Resolutionslemma(3.3.6): $\mathcal{K}$ und $Res(\mathcal{K})$ sind äquivalent.
<u>Induktion über n:</u> $\mathcal{K}$ und $Res^n(\mathcal{K})$ sind äquivalent. $\Box \in Res^*(\mathcal{K})$.
$\Rightarrow$ Es existiert ein $n \in \mathbb{N}$ mit $\Box \in Res^n(\mathcal{K})$
$\Rightarrow Res^n(\mathcal{K})$ ist unerfüllbar
$\mathcal{K}$ ist unerfüllbar (da $\mathcal{K} \equiv Res^n(\mathcal{K})$)

„$\Rightarrow$" (Vollständigkeit)
$\mathcal{K}^+, \mathcal{K}^-$ enthält die atomare Formel $\mathcal{A}$ nicht mehr.
$\mathcal{K}^+, \mathcal{K}^-$ sind beide unerfüllbar.
<u>Ann.:</u> $\mathcal{K}^+$ nicht erfüllbar.
$\Rightarrow S \models \mathcal{K}^+$ für eine Struktur S.
erweitere S, so dass $S \models \mathcal{A}$ Dann gilt $S \models \mathcal{K}^+$.
Analog $\mathcal{K}^-$ unerfüllbar.
Induktionshypothese: $\Box \in Res^*(\mathcal{K}^+)$, $\Box \in Res^*(\mathcal{K}^-)$
Es existieren Folgen von Klauseln $K_1, \ldots, K_n$ mit $K_1 = \Box$, so dass für alle $1 \leq i \leq m$ gilt:

- $K_i \in \mathcal{K}^+$ oder

- K_i ist Resolvent von K_j, K_k, j,k<i

Falls alle im Resolutionsbeweis benängeln Klauseln aus $\mathcal{K}^+$ auch schon in $\mathcal{K}'$ sind, dann gilt: $\Box \in Res^*$. In diesem Fall werden also nur Klauseln aus $\mathcal{K}^-$ benutzt die $\neg A$ nicht enthalten.
D.h. $\{\neg A\} \in Res^*(\mathcal{K}')$

Analog: $\Box \in Res^*(\mathcal{K}^-) \Rightarrow \Box \in Res^*(\mathcal{K}^-)$ oder $\{A\} \in Res^*(\mathcal{K}^-)$
Falls $\{A\}$, $\{\neg A\} \in Res^*(\mathcal{K}')$, dann auch $\Box \in Res^*(\mathcal{K}')$

Grundresolutionsalgorithmus

1. Sei ψ die Formel $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n \neg\varphi$

2. Überführe ψ in SNF $\forall X_1, \ldots, X_n \xi$

3. Überführe ξ in KNF bwz. in die entsprechende Klauselmenge $\mathcal{K}(\xi)$

4. Berechne $Res^*(\mathcal{K})$. Falls leere Klausel gefunden, brich mit YES ab.

Beispiel 3.3.8

Zeige Unerfüllbarkeit von $\psi = \forall X, Y((\neg p(X) \vee \neg p(f(a)) \vee aY) \wedge p(Y) \wedge (\neg p(g(b, X)) \vee q(b)))$
$\mathcal{K}(\psi) = \{\{\neg p(X), \neg p(f(a)), aY\}, \{p(Y)\}, \{\neg p(g(b, X)), q(b)\}\}$
Aufzählung: $\{K_1, K_2, \ldots\}$
$K_1 = \sigma_1(\{\neg p(X), \neg p(f(a)), aY\})$ mit $\sigma_1 = [X/f(a), Y/b]$
$K_2 = \sigma_2(\{p(X)\})$
$K_3 = \sigma_3(\{p(Y)\})$

Satz 3.3.9 (Korr.+Vollst. des Grundresolutionsalg.)

(a) Falls eine (endl.) Klauselmengen $\mathcal{K}$ unerfüllbar ist, dann existiert eine endliche Menge von Grundinstanzen von Klauslen aus $\mathcal{K}$, d.h. eine endliche Teilmenge von $\{\sigma(K) \mid K \in \mathcal{K}, \sigma$ Grundsubstitution$\}$ die ebenfalls unerfüllbar ist.

(b) Sei $\forall X_1, \ldots, X_n \psi$ in SNF, ψ KNF. Dann ist $\forall X_1, \ldots, X_n \psi$ unerfüllbar $\Leftrightarrow$ es exisiert eine Folge von Klauslen $K_1, \ldots, K_n$ mit $K_n = \Box$, so dass für alle $1 \leq i \leq m$:

- K_i ist Grundinstanz einer Klausel aus $\mathcal{K}$ oder
- K_i ist Resolvent von K_j, K_k mit j,k<i

Beweis:

„(a)“: Sei $\mathcal{K} = \{K_1, \ldots, K_i\}$. Sei ψ_i die nicht allquantifizierte Disjunktion der Literale aus K_i, sei $\psi = \psi_1 \wedge \ldots \wedge \psi_r$.
$\Rightarrow \mathcal{K}$ entspricht der Formel $\forall X_1, \ldots, X_n \psi$
Daher:
$\mathcal{K}$ ist unerfüllbar
$\Leftrightarrow \forall X_1, \ldots, X_n \psi$ ist unerfüllbar
$\Leftrightarrow = E(\forall X_1, \ldots, X_n \psi) = \{sigma(\psi) | \sigma$ ist Grundsubstitution$\}$ ist unerfüllbar
$\Leftrightarrow \{\sigma(K) | K \in \mathcal{K}, \sigma$ Grundsubstitution$\}$ ist unerfüllbar.

„(b)“: $\forall X_1, \ldots, X_n \xi$ unerfüllbar
$\Leftrightarrow$ es existiert endliche Teilmenge von $\{\sigma(K) | K \in \mathcal{K}(\forall X_1, \ldots, X_n \xi)\}$ die unerfüllbar ist (wie in (a)).
$\Leftrightarrow \Box$ ist durch Resolution aus einer dieser endlichen Megen herleitbar.

3.4 Prädikatenlogische Resolution

Nachteil der Grundresolution: Man muss *geschickte* Grundinstanzen der Variablen wählen, damit man damit nicht nur den nächsten, sondern auch später Resolutionsschritte ermöglicht.
Verbesserung: Verwende keine Grundsubstitution, sondern *zurückhaltende* beliebige Substitutionen. Instanziere nur soweit wie es für den nächsten Resolutionsschritt nötig ist, lasse ansonsten Variablen stehen.

Beispiel 3.4.1

$\{\underbrace{\{p(X), \neg q(X)\}}_{K_1}, \underbrace{\{\neg p(f(Y))\}}_{K_2}, \underbrace{\{q(f(a))\}}_{K_3}\}$.
Um K_1 und K_2 zu resolvieren, müssen X und Y instanziiert werden. Damit nachher der entstehende Resolvent mit K_3 resolviert werden kann, muss man schon bei Resolution von K_1, K_2 eine *geschickte* Instanziierung wählen, wenn man nur Grundsubstitution erlaubt: [X/f(a),Y/a].
Besser: erlaube Instanziierung mit Nicht-Grundtermen damit p(X) und ¬p(f(Y)) komplementär werden können. Suchtman einen Unifikator vn $\{p(X), p(f(X))\}$. Allgemeiner Unifikator ist $\{X/f(Y)\}$. Resolvent ist dann $\{\neg q(f(Y))\}$.

2.5.2006

Definition 3.4.2 (Unifikation)

Eine Klausel $K = \{L_1, \ldots, L_n\}$ heißt unifizierbar $\Leftrightarrow$ eine Substitution σ existiert mit $\sigma(L_1) = \ldots = \sigma(L_n)$ (d.h. $\mid \sigma(K) \mid= 1$). Eine solche Substitution heißt Unifikator von K. σ heißt allgemeinster Unfikator (most general unifier, mgu), falls es für jeden Unifikator σ' eine Substitution δ mit $\sigma' = \delta \circ \sigma$.
Falls eine Klausel unifizierbar ist, dann hat sie einen mgu, dieser ist bis auch Variablenumbenennung eindeutig.

3.4.3 (Unifikationsalgorithmus von Robinson 1965)

1. Sei $\sigma = \emptyset$ $(\sigma = id)$
2. Falls $\mid \sigma(K) \mid= 1$, brich ab, gib mgu σ aus
3. Durchsuche alle $\sigma(L_i)$ parallel von links nach rechts, bis in zwei Literalen die gelesenen Zeichen verhschieden sind.
4. Falls keines dieser beiden Zeichen einen Variable ist, brich mit clash failure ab.
5. Sonst sei X Variable in einem Literal und t der Teilterm im anderen Literal. Falls X in t vorkommt, dann brich mit Occur-Failure ab.
6. Setze sonst $\sigma = \{X/t\} \circ \sigma$ und gehe zu 2

Beispiel 3.4.3

- $\{p(f(X,Y), p(g(X,Y))\} \Rightarrow$ Clash-Failure
- $\{p(X), p(h(X))\} \Rightarrow$ Occur-Failure
- $\{\neg p(f(Z), g(a,Y)), h(Z), \neg p(f(f(U,V),W), h(f(a,Y)))\}$
  $\Rightarrow \sigma = \{Z/f(U,V)\}$
  $\sigma(K) = \{\neg p(f(f(U,V), \underline{g(a,Y)}), h(f(U,V))), \neg p(f(f(U,V), \underline{W}), h(f(a,Y)))\}$
  $\sigma = \{W/g(a,Y)\} \circ \{Z/f(U,V)\} = \{W/g(a,Y), Z/f(U,V)\}$
  $\sigma(K) = \{\neg p(f(f(U,V), g(a,Y)), \underline{h(f(U,V))}), \neg p(f(f(U,V), g(a,Y)), \underline{h(f(a,Y))})\}$
  $\sigma = \{U/a, W/g(a,Y), Z/f(U,V)\}$
  $\sigma(K) = \{\neg p(f(f(a,V), g(a,Y)), \underline{h(f(a,V))}), \neg p(f(f(a,V), g(a,Y)), \underline{h(f(a,Y))})\}$
  $\sigma = \{Y/V, U/a, W/g(a,Y), Z/f(U,V)\}$ ist der mgu!
  $\Rightarrow |\sigma(K)| = 1$

Satz 3.4.4 (Terminierung und Korrektheit des Unifikationsalg.)

Der Unifikationsalgorithmus terminiert für jede Klausel K und ist korrekt, d.h. er liefert einen mgu für K $\Leftrightarrow$ K ist unifizierbar.
Beweis: Terminierung folgt, da in jedem Schleifendurchlauf die Anzahl der Variablen in $\sigma(K)$ um 1 abnimmt. Falls Algorithmus mit Erfolg terminiert und Substitution σ ausgibt, so gilt $|\sigma(K)| = 1$, d.h. σ ist Unifikator von K. Es bleibt zu zeigen: K unifizierbar $\Rightarrow$ Algorithmus terminiert mit Erfolg und σ ist allgemeiner Unfikator.

Sei n die Anzahl der Schleifendurchläufe bei der Eingabe der Klausel K. Für alle $0 \leq i \leq n$ sei σ_i der Wert von σ anch dem i-ten Schleifendurchlauf.
Wir zeigen folgendes für alle $0 \leq i \leq n$:
Für jeden Unifikator σ' von K gilt: $\sigma' = \sigma' \circ \sigma_i$ (*)
Aus (*) folgt:

- Algorithmus bricht nicht mit Fehlschlag ab.
  Sonst: $\sigma_n(K)$ wäre nicht unifizierbar
  Aber: K hat einen Unifikator σ', $1 = |\sigma'(K)| = \sigma'(\sigma_n(K))| \Rightarrow \sigma'$ ist Unifikator von $\sigma_n(K)$.

- Also bricht Algorithmus mit Erfolg ab $\Rightarrow |\sigma_n(K)| = 1$, d.h. σ_n ist Unifikator von K.

Dann ist σ_n auch mgu, denn für alle Unifikatoren σ' existiert ein δ mit $\sigma = \delta \circ \sigma_n$. Zeig jetzt (*) durch Induktion über i:

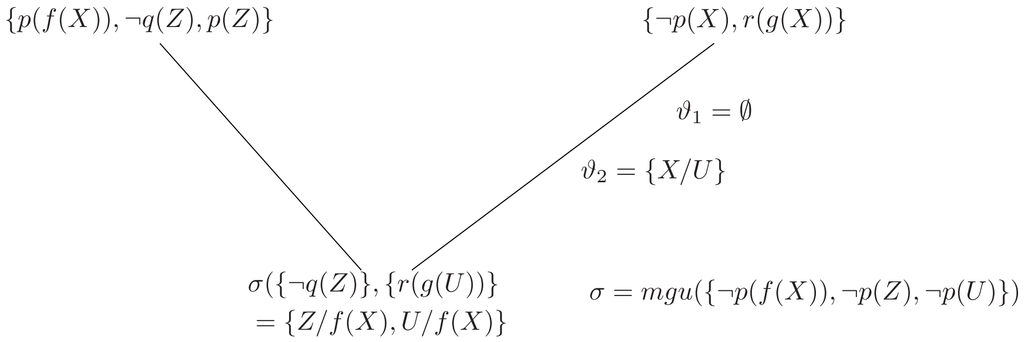
<u>IA:</u> i=0: $\sigma_0 = \emptyset$ (Identität)
<u>IS:</u> i >0: $\sigma_i = \{X/t\} \circ \sigma_{i-1}$, wobei zur Unifikation von $\sigma_{i-1}(K)$ X und t "gleich gemacht werden müssen"
<u>Ind.Hyp.:</u> $\sigma' = \sigma \circ \sigma_{i-1}$, z.z. $\sigma' = \sigma' \circ \sigma_i$
$\sigma' \circ \sigma_i = \sigma' \circ \{X/t\} \circ \sigma_{i-1} = \sigma' \circ \sigma_{i-1} \underbrace{=}_{Ind.Hyp.} \sigma'$
σ' ist Unifikator von $\sigma_{i-1}(K)$, denn $1 = |\sigma'(K)|$. Daher: $\sigma'(K) = \sigma(t) = |\sigma'(\sigma_{i-1}(K))|$
$\Rightarrow \sigma' = \sigma' \circ \{X/t\}$. *q.e.d.*

Definition 3.4.5 (Prädikatenlogische Resolution)

Seien K_1, K_2 Klauseln, dann ist die Klausel R <u>Resolvent</u> von $K_1, K_2 \Leftrightarrow$

- Es gibt Variablenumbenennungen ϑ_1, ϑ_2, so dass $\vartheta_1(K_1)$ und $\vartheta_2(K_2)$ keine gemeinsamen Variablen enthalten.
- Es gibt Literale $L_1, \dots, L_n \in \vartheta_1(K_1)$ und $L'_1, \dots, L'_n \in \vartheta_2(K_2)$ mit $n, m \geq 1$, so dass $\{\bar{L}_1, \dots \bar{L}_m, L_1, \dots, L_n\}$ mit einem σ unifizierbar sind.
- $R = \sigma((\vartheta_1(K_1) \setminus \{L_1, \dots, L_m\}) \cup (\vartheta_2(K_2) \setminus \{L'_1, \dots, L'_n\}))$
- $Res(\mathcal{K}) = \mathcal{K} \cup \{R \mid \text{R ist Resolvent von } \mathcal{K}\}$
- $Res^*(\mathcal{K})$ analog.

Beispiel 3.4.6



5.5.2006

Lemma 3.4.7 (Prädikatenlogisches Resolutionslemma)

Sei $\mathcal{K}$ eine Menge von Klauseln, $K_1, K_2 \in \mathcal{K}$, R ist Resolvent von K_1, K_2. Dann sind $\mathcal{K}$ und $\mathcal{K} \cup \{R\}$ äquivalent.
<u>Beweis:</u>
$\mathcal{K} \cup \{R\} \models \mathcal{K}$ trivial.

z.z.: $\mathcal{K} \models \mathcal{K} \cup \{R\}$, d.h. $\mathcal{K} \models R$ Sei $S \models \mathcal{K}$. z.z.: $S \models R$.
$\vartheta_1(K_1) = \{L_1, \ldots, L_m, L_{m+1}, \ldots, L_p\}$
$\vartheta_2(K_2) = \{L'_1, \ldots, L'_n, L'_{n+1}, \ldots, L_q\}$
mit $p \geq m$, $q \geq n$.
$R = \sigma(\{L_{m+1}, \ldots, L_p, L_{n+1}, \ldots, L_q\})$
$\sigma(L_1) = \ldots = \sigma(L_m) = \bar{L}$
$\sigma(L'_1) = \ldots = \sigma(L'_n) = L$
$S \models K_1$, $S \models K_2$
<u>Annahme:</u>
$S \not\models R$. Es existiert also $S \not\models \forall \ldots \sigma(L_{m+1} \vee \ldots \vee L_p \vee L'_{n+1} \vee \ldots \vee L'_q)$
Sei $S = (\mathcal{A}, \alpha)$, dann existieren Variablenbelegung β und $I = (\mathcal{A}, \alpha, \beta)$ mit
$I \not\models \sigma(L_{m+1} \vee \ldots \vee L_p \vee L'_{n+1} \vee \ldots \vee L'_q)$.
Sei $\sigma\{X_1/t_1, \ldots, X_n/t_n\}$, $I' = (\mathcal{A}, \alpha, \beta[\![X_1/I(t_1), \ldots, X_k/I(t_k)]\!])$ (*Subtitutionslemma 2.2.3*).
$I' \not\models (L_{m+1} \vee \ldots \vee L_p \vee L'_{n+1} \vee \ldots \vee L'_q)$ (*)
Aus $S \models K_1$ und $S \models K_2$ folgt $S \models \vartheta_1(K_1)$, $S \models \vartheta_2(K_2)$ (Variablen sind implizit allquantifiziert)
<u>Daher:</u>
$I' \models L_{m+1} \vee \ldots \vee L_p \vee L'_{n+1} \vee \ldots \vee L'_q \overset{(*)}{\Rightarrow} I' \models L_1 \vee \ldots \vee L_m$
$I' \models L'_1 \vee \ldots \vee L'_n \vee L'_{n+1} \vee \ldots L'_q \overset{(*)}{\Rightarrow} I' \models L'_1 \vee \ldots \vee L'_n$
$\overset{(Subst.Lemma)}{\Rightarrow} I \models \sigma(L_1) \vee \ldots \sigma(L_m) \Rightarrow I \models \bar{L}$
$I \models \sigma(L'_1) \vee \ldots \sigma(L'_n) \Rightarrow I \models L$ Widerspruch!

Aus Resolutionslemma folgt sofort die Korrektheit:
$\square \in Res^*(\mathcal{K}) \Rightarrow$ es existiert n mit $\square \in Res^n(\mathcal{K})$ **Aber** $\mathcal{K}$ ist äquivalent zur $Res^n(\mathcal{K})$.
<u>Zeige nun Vollständigkeit der präd-log. Resolution</u>

Lemma 3.4.8 (Lifting Lemma)

Seien K_1, K_2 zwei Klauseln mit Grundinstanzen K'_1, K'_2. Falls R' Resolvent von K'_1, K'_2 ist, dann existiert Resolvent R von K_1, K_2, so dass R' Grundinstanz von R ist.

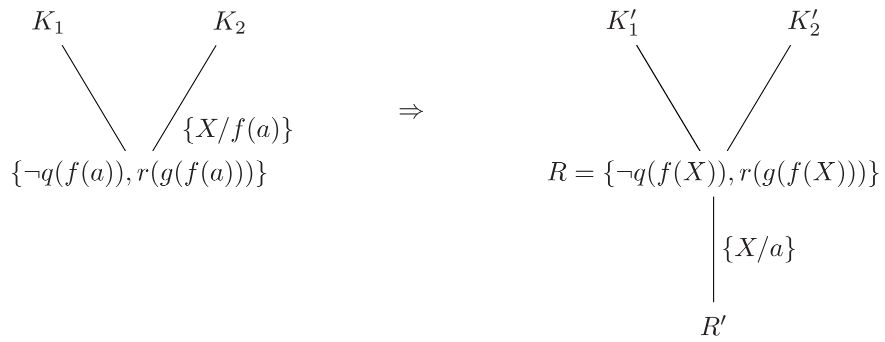
$$\begin{tikzcd}[arrows=dash]
K_1 \arrow[d] & & K_2 \arrow[d] \\
K'_1 \arrow[dr] & & K'_2 \arrow[dl] \\
& R &
\end{tikzcd}
\quad \Rightarrow \quad
\begin{tikzcd}[arrows=dash]
K_1 \arrow[dr] & & K_2 \arrow[dl] \\
& R \arrow[d] & \\
& R' &
\end{tikzcd}$$

<u>Beweis:</u>
Seien ϑ_1, ϑ_2 Variablenumbenennugen, so dass $\vartheta_1(K_1), \vartheta_2(K_2)$ keine gemeinsamen Variablen haben. Es exisitiert also eine <u>gemeinsame</u> Grundsubstitution σ mit $\sigma(\vartheta_1(K_1)) = K'_1$, $\sigma(\vartheta_2(K_2)) = K'_2$.
Es existiert also $L \in K'_1$, $\bar{L} \in K'_2$ mit $R' = (K'_1 \setminus \{L\}) \cup (K'_2 \setminus \{\bar{L}\})$. Seien $L_1, \ldots, L_m \in \vartheta_1(K_1)$ <u>alle</u> Urlbilder von L unter σ ($\sigma(L_1) = \ldots = \sigma(L_m) = L$), seien $L'_1, \ldots, L'_n \in \vartheta_2(K_2)$ <u>alle</u> Urlbilder von $\bar{L}$ unter σ ($\sigma(L'_1) = \ldots = \sigma(L_n) = \bar{L}$).
Da σ Unifikator von $\{\bar{L}_1, \ldots, \bar{L}_m, L'_1, \ldots, L'_n\}$ ist, existiert auch ein mgu σ' (*)
$\Rightarrow K_1$ und K_2 haben Resolventen $R = \sigma'((\vartheta_1(K_1) \setminus \{L_1, \ldots, L_m\}) \cup (\vartheta_2(K_2) \setminus \{L'_1, \ldots, L'_n\}$
z.z.: R' ist Grundinstanz von R.
Wegen (*) existiert eine Substitution δ mit $\sigma = \delta \circ \sigma'$, δ ist Grundsubstitution.
Zeige nun: $R' = \delta(R)$
$R' = (K'_1 \setminus \{L\}) \cup (K'_2 \setminus \{\bar{L}\})$

$= (\sigma(\vartheta_1(K_1)) \setminus \{L\}) \cup (\sigma(\vartheta_2(K_2)) \setminus \{\bar{L}\})$
$= \sigma((\vartheta_1(K_1) \setminus \{L_1, \ldots, L_m\}) \cup (\vartheta_2(K_2) \setminus \{L'_1, \ldots, L'_n\}))$
$= \delta(\sigma'((\vartheta_1(K_1) \setminus \{L_1, \ldots, L_m\}) \cup (\vartheta_2(K_2) \setminus \{L'_1, \ldots, L'_n\})))$
$= \delta(R)$ *q.e.d.*

Beispiel 3.4.9

$K_1 = \{p(f(X)), \neg q(Z), p(Z)\}$, $K_2 = \{\neg p(X), r(g(X))\}$
$K'_1 = \{p(f(a)), \neg q(f(a))\}$, $K'_2 = \{\neg p(f(a)), r(g(f(a)))\}$



Satz 3.4.10 (Korr. + Vollst. der prädikatenlogischen Resolution)

Sei $\mathcal{K}$ Menge von Klauseln. Dann gilt: $\mathcal{K}$ unerfüllbar gdw. $\Box \in Res^*(\mathcal{K})$.
<u>Beweis:</u>
„$\Leftarrow$ (Korrektheit)
bereits gezeigt, folgt aus Resolutionslemma *3.4.7*
„$\Rightarrow$“(Vollständigkeit) $\mathcal{K}$ unerfüllbar
$\Rightarrow$ es existiert eine endliche unerfüllbare Mengen von Grundinstanzen von Klauseln aus $\mathcal{K}$. (*Satz 3.3.9(a)*)
Rightarrow Wegen Vollständigkeit der aussagenlogischen Resolution (*Satz 3.3.7*) existiert Folge $K'_1, \ldots, K'_m$ mit $K'_m = \Box$ und für alle $1 \leq i \leq m$:

- K'_i ist Grundinstanz einer Klausel aus $\mathcal{K}$ oder
- K'_i ist Resolvent von K'_j, K'_kmit j, k<i

Mit dem Lifting-Lemma *3.4.8* erzeugen wir Folge von Klauseln $K_1, \ldots, K_m$, so dass K'_i Grundinstanz von K_i ist und $K_i \in Res^*(\mathcal{K})$:

- Falls K'_i Grundinstanz von $K \in \mathcal{K}$, dann $K_i = K$
- Falls K'_i Resolvent von K'_j, K'_k ist (j, k<i), dann existiert bereits $K_j, K_k \in Res^*(\mathcal{K})$, so dass K'_j Grundinstanz von K_j ist und K'_k von K_k. Lifing Lemma: K_j und K_k haben Resolventen K_i, so dass K'_i Grundinstanz von K_i ist $\Rightarrow K_i \in Res^*(\mathcal{K})$

$\Rightarrow K_m \in Res^*(\mathcal{K})$, $\underbrace{K'_m}_{\Box}$ ist Grundinstanz von K_m

$\Rightarrow K_m = \Box \in Res^*(\mathcal{K})$. *q.e.d.*

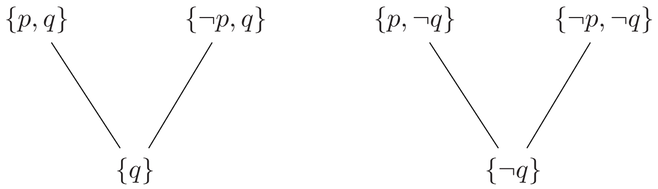
10.5.2006
Prädikatenlogische Resolution ist korrekt und vollständig.
$\mathcal{K}$ unerfüllbar$\Leftrightarrow \Box \in Res^*(\mathcal{K})$
Nachteil: $Res^*(\mathcal{K})$ ist zu groß, da man beliebig Klauseln miteinander resolvieren darf.
Lösung: Schränke Resolution ein, so dass man nur noch zwischen bestimmten Klauseln resolvieren darf, und so dass die Vollständigkeit erhalten bleibt.$\rightarrow$ effizienter und genauso mächtig.

3.5 Entschränkung der Resolution

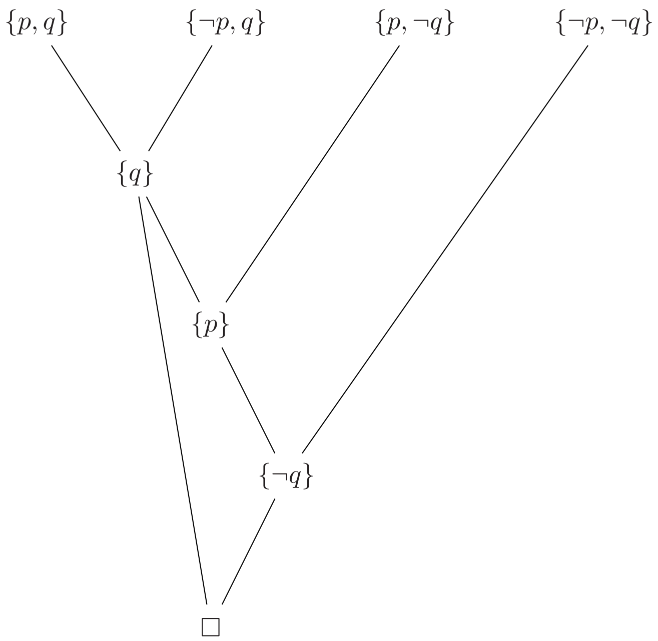
Definition 3.5.1 (Lineare Resolution)

Sei $\mathcal{K}$ eine Klauselmenge. Die leere Klausel $\square$ ist aus der Klausel $k \in \mathcal{K}$ linear resolvierbar gdw. es eine Folge von Klauseln $K_1, ..., K_n$ gibt mit $K_1 = K$, $K_n = \square$ so dass für alle. $2 \leq i \leq m$ gilt: K_i ist Resolvent von K_{i-1} und einer Klausel aus $\{K_1, ..., K_{i-1}\} \bigcup \mathcal{K}$.

Beispiel 3.5.2



$p, q \in \Delta_0$. Kein linearer Resolutionsbeweis, denn im 2. Schritt hätte man mit $\{q\}$ und einer anderen Klausel resolvieren müssen.



Dies ist ein linearer Resolutionsbeweis.

Satz 3.5.3 (Korr. + Vollst. der linearen Resolution)

Sei $\mathcal{K}$ eine Klauselmenge. Dann ist $\mathcal{K}$ unerfüllbar $\Leftrightarrow$ $\square$ aus einer Klausel in $\mathcal{K}$ linear resolvierbar ist. Falls $\mathcal{K}$ eine minimale unerfüllbare Klauselmange ist(d.h. $\mathcal{K} \setminus K$ ist erfüllbar für alle $k \in \mathcal{K}$), dann ist $\square$ sogar aus jeder Klausel in $\mathcal{K}$ linear resolvierbar.
<u>Beweis:</u>
Korrektheit (“$\Leftarrow$“) folgt aus Korrektheit der (vollen) Resolution, (Satz 3.4.10). Vollständigkeit (“$\Rightarrow$“): Zeige erst Vollständigkeit d. lin.Res in der Aussagenlogik + lifte dies in die Prädikatenlogik.

1. Vollständigkleit der aussagenlogische lineare Resolution (d.h. $\mathcal{K}$ ist variablenfrei)
   Sei $\mathcal{K}_{min} \subseteq \mathcal{K}$ eine minimale unerfüllbare Teilmenge von $\mathcal{K}$. Offensichtlich gilt $\mathcal{K}_{min} \neq \emptyset$ ($\emptyset$ ist erfüllbar, sogar allgemeingültig). Wir zeigen, dass $\Box$ aus jeder Klausel K in $\mathcal{K}_{min}$ linear resolvierbar ist. Induktion über die Anzahl in der verschiedenen atomaren Formeln in $\mathcal{K}_{min}$.
   <u>IA:</u> $n = 0 \Rightarrow k = \Box$
   <u>IS:</u> n>0
   1.Fall: $|K| = 1$, d.h. $K = \{L\}$
   2. Fall: $|K|$>1 (Übung)
   $\mathcal{K}^+$ entsteht aus $\mathcal{K}_{min}$,

   - indem man alle Klauseln wegläßt, die L enthalten
   - und $\bar{L}$ aus den verbleibenden Klauseln streicht.

   $\Rightarrow \mathcal{K}^+$ enthält höchstens $n-1$ verschiedene atomare Formeln. $\mathcal{K}^+$ ist unerfüllbar
   ($S \models \mathcal{K}^+ \Rightarrow S' \models \mathcal{K}_{min}$ Widerspruch, da $\mathcal{K}_{min}$ unerfüllbar.)
   $\Rightarrow \Box$ ist aus jeder Klausel in $\mathcal{K}^+$ linear resolvierbar(*).
   Es muss eine Klausel $K^+ \in \mathcal{K}^+$ geben, so dass $K^+ \notin \mathcal{K}_{min}$ (aber $K^+ \cup \{\bar{L}\} \in \mathcal{K}_{min}$)
   Wegen (*) ist $\Box$ aus K^+ in $\mathcal{K}^+$ lin.resolvierbar.
   Es gibt also eine Folge von Klauseln $K_1, ...K_m$ mit $K_1 = K^+$, $K_m = \Box$ und K_i ist Resolvent von K_{i-1} und einer Klausel aus $\{K_1, ..., K_{i-1} \bigcup \mathcal{K}^+$. Für alle $1 \leq i \leq m$ ex. dann auch eine lineare Resolutionsfolge $K, \ldots, K_i$ in $\mathcal{K}_{min}$.
   Induktion über i:
   <u>IA:</u> $i = 1$

   $$\begin{array}{ccc} K = \{L\} & & K^+ \cup \{\bar{L}\} \\ & \diagdown \quad \diagup & \\ & K^+ = K_1 & \end{array}$$

   <u>IS:</u> i>1
   K_i ist Resolvent von K_{i-1} und Klausel aus $\{K_1, ..., K_{i-1}\} \cup \mathcal{K}^+$ IndHyp.:
   $K, \ldots, K_1, \ldots, K_2, \ldots, K_{i-1}$ ist linear resolvierbar in $\mathcal{K}_{min}$.

   - falls K_i Resolvent von K_{min} und $K' \in \{K_1, ..., K_{i-1}\}$ ist, dann ist auch $K, K_1, \ldots, K_2, \ldots, K_{i-1}, K_i$ linear resolvierbar in $\mathcal{K}_{min}$
   - Falls K_i Resolvent von K_{i-1} und $K' \in \mathcal{K}^+ \cap \mathcal{K}_{min}$, dann ist auch $K, K_1, \ldots, K_2, \ldots, K_{i-1}, K_i$ linear resolvierbar in $\mathcal{K}_{min}$
   - Falls K_i Resolvent von K_{i-1} und $K' \in \mathcal{K}^+, K' \notin \mathcal{K}_{min}$, dann ist $K' \cup \{\bar{L}\} \in \mathcal{K}_{min}$

   Dann ist $K, K_1, \ldots, K_2, \ldots, K_{i-1}, K_i \cup \{\bar{L}\}, K_i$

2. Vollständigkeit der prädikatenlogischen linearen Resolution
   analog zum Vollständigkeitsbeweis der Vollen prädikatenlogischen Resolution.
   $\mathcal{K}$ unerfüllbar $\Rightarrow$ es existiert endliche unerfüllbare Menge von Grundinstanzen von Klauseln aus $\mathcal{K}$ *(Satz 3.3.5(a))* Vollständigkeit der linearen aussagenlosischen Resolution $\Rightarrow$ es exisiter Herleitung von $\Box$ durch lineare Resolution aus der Menge der Grundinstanzen der Klauseln von $\mathcal{K}$
   Lifting Lemma $\Rightarrow$ es existiert Herleitung von $\Box$ durch lineare Resolution aus $\mathcal{K}$ (genauso wie im Beweis von *Satz 3.4.10*)
   <u>Bisher:</u> Bei jedem Resolutionsschritt liegt eine Elternklausel fest (zuletzt erzeugter Resolvent). Andere Elternklausseln ist aus ursprünglicher Klauselmenge (oder ein früher erzeugter Resolvent) $\Rightarrow$ Input Resolution.

Definition 3.5.4 (Input-Resolution)

Sei $\mathcal{K}$ eine Klauselmenge. Die leere Klausel $\Box$ ist aus der Klausel $K \in \mathcal{K}$ durch Input-Resolution herleitbar $\Leftrightarrow$ Folge von Klauseln $K_1, \ldots, K_m$ existiert mit $K_1 = K$, $K_m = \Box$ und so dass für alle $2 \leq i \leq m$ gilt: K_i ist Resolvent von K_{i-1} und einer Klausel aus $\mathcal{K}$.
$\Rightarrow$ Input-Resolution ist Spezialfall der linearen Resolution.

Definition 3.5.6 (Horn-Klausel)

Eine Klausel K ist eine Hornklausel $\Leftrightarrow$ sie enthält höchstens ein positives Literal.
Eine Hornklausel $\{\neg A_1, \ldots, \neg A_k\}$ ohne pos. Literalte hei"st negativ. Eine Hornklausel $\{B, \ldots, \neg A_1, \ldots, \neg A_k\}$ mit positivem Literal heißt definit.

12.5.2006

- $\mathcal{K}$ unerfüllbar gdw. durch Resolution herleitbar
- $\mathcal{K}$ unerfüllbar gdw. durch lin. Resolution herleitbar
- $\mathcal{K}$ unerfüllbar gdw. durch Input Resolution herleitbar, falls $\mathcal{K}$ nur Hornklausen enthält

Hornklauseln entspr. Klauseln mit höchstens einem pos. Literal. Hornklauseln entspr. Implikationen, wobei links von „$\rightarrow$“ eine Konjunktion steht

Beispiel:

$\{\{p, \neg q\}, \{\neg r, \neg p, s\}, \{s\}\}$
bedeutet $(q \rightarrow p) \wedge (r \wedge p \rightarrow s) \wedge s$

Zusammehang zur Logikprogrammierung:

- Fakten entsprechen Hornklauseln ohne negative Literale, z.B. $\{s\}$
  Schreibweise `s.`

- Regeln entsprechen Hornklausel mit positiven und negativen Literalen,
  z.B. $\{\neg r, \neg p, s\}$
  Schreibweise: `S:-r,p`

- Anfragen entsprechen Hornklauseln ohne positive Literale,
  z.B. $\{\neg p, \neg q\}$
  Schreibweise: `?-p,q.`

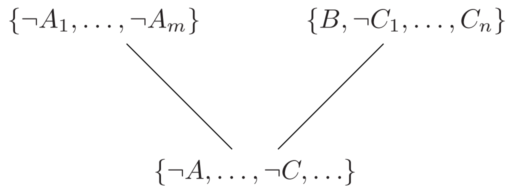
Definite Hornklauseln entsprechen den Klauseln des Logikprogramms.
Negative Hornklauseln entsprechen der Anfrage an das Logikprogramms.
Betrachtung von Hornklauseln schränkt die Ausdrucksstärke ein, aber Unerfüllbarkeitsnachweis ist viel effizienter.

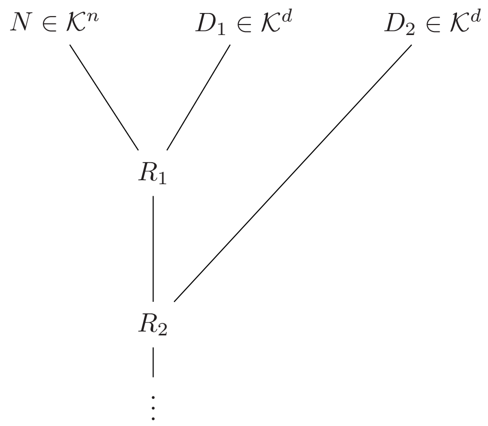
	Volle Klausel	Hornklausel
AL	NP-Vollständig (SAT)	Linearzeit
FO-Logik	unerfüllbar, aber semi-entscheidbar	unerfüllbar, aber semi-entscheidbar (effizienter)

Statt Vollständigkeit der Input-Resolution auf Hornklauselmengen nachzuweisen, schränke Input-Resolution weiter ein zur SLD Resolution. Dann zeigen wir, dass SLD-Resolution auf Hornklauselmengen vollständig ist.

Definition 3.5.7 (SLD-Resolution)

Sei $\mathcal{K}$ eine Menge von Hornklauseln mit $\mathcal{K} = \mathcal{K}^d \uplus \mathcal{K}^n$, wobei: $\mathcal{K}^d$ die definiten Klauseln und $\mathcal{K}^n$ die negativen Klauseln von $\mathcal{K}$ enthält. Die leere Klausel $\Box$ ist aus $K \in \mathcal{K}^n$ durch SLD-Resolution herleitbar $\Leftrightarrow$ eine Folge $K_1, K_2, \ldots, K_m$ existiert mit $K_1 = K \in \mathcal{K}^n$, $K_m = \Box$, für alle $2 \leq i \leq m$: K_i ist Resolvent von K_{i-1} und. einer Klausel aus $\mathcal{K}$. ($K_1, \ldots, K_m$ sind immer negative Klauseln)
Negative Hornklauseln können nur mit definiten Hornklauseln resoviert werden.





SLD = **L**inear resolution with **S**election function for **D**efinite Clauses.
Hierbei handelt es sich vielleicht doch um eine handeslübliche Droge ;)

2 Indeterminismen:

1. Wahl des Literal aus der neg. Klausel, mit dem resolviert werden soll.
2. Wahl der andere definite Elternklausel

Selektionsfunktion "löst" diese Indeterminismen.
Bisher: Ignoriere Selektionsfunktion $\Rightarrow$ "LUSH- Resolution"

LUSH = **L**inerar resolution with **U**nrestricted **S**election for **H**orn clauses

Satz 3.5.8 (Korr. + Vollst. der SLD-Resolution)

Sei $\mathcal{K}$ eine Menge von Hornklauseln. Dann ist $\mathcal{K}$ unerfüllbar $\Leftrightarrow$ $\Box$ aus einer negativen Klausel $N \in \mathcal{K}$ durch SLD-Resolution herleitbar ist.
Beweis:
Korrektheit klar (da SLD-Resolution Spezialfall der vollen Resolution ist, die korrekt ist (*Satz 3.4.10*))

Vollständigkeit ("⇒") $\mathcal{K}$ enthählt eine negative Klausel, denn jede Menge definiter Hornklauseln ist erfüllbar (Ein Modell ist die Struktur, die alle atomaren Formeln erfüllt.)
Sei $\mathcal{K}_{min} \subseteq \mathcal{K}$ eine minimale unerfüllbare Teilmenge. $\Rightarrow$ $\mathcal{K}_{min}$ enthält auch negative Klausel N. Wegen der Vollständigkeit der linearen Resolution *(Satz 3.5.3)* ist $\Box$ aus jeder Klausel von $\mathcal{K}_{min}$ durch lineare Resolution herleitbar. Also existier auch ein lineare Resolutionsbeweis, von $\Box$, der mit N startet. Dieser lineare Resolutionsbeweis ist auch ein SLD-Resolutions Beweis:

1. startet mit negativer Klausel
2. da alle Resolventen negativ sind, können nie 2 Resolventen miteinander resolviert werden $\Rightarrow$ ist Input- Resolution. *qed*

Weitere letzte Einschränkung:
Binäre Resolution: $m = n = 1$, d.h. in jedem Resolutionsschritt wird nur zwischen jeweils einem Literal der Elternklauseln resolviert. Aber binäre Resolution ist nicht vollständig.

Beispiel 3.5.9

$$\{p(X), p(Y)\} \quad \{\neg p(U), \neg p(V)\}$$

$$\sigma = \{X/V, Y/V, U/V\}$$

$$\Box$$

Aber: Binäre Resolution ist vollständig auf Hornklauselmengen.

$$\{\underline{p(X)}, p(Y)\} \quad \{\underline{\neg p(U)}, \neg p(V)\}$$

$$\sigma = \{X/U\}$$

$$\{p(Y), \neg p(V)\}$$

Dieser Resolutionsbeweis kommt offenbar nie zu $\Box$.

Satz 3.5.10 (Korr. + Vollst der binären SLD-Resolution)

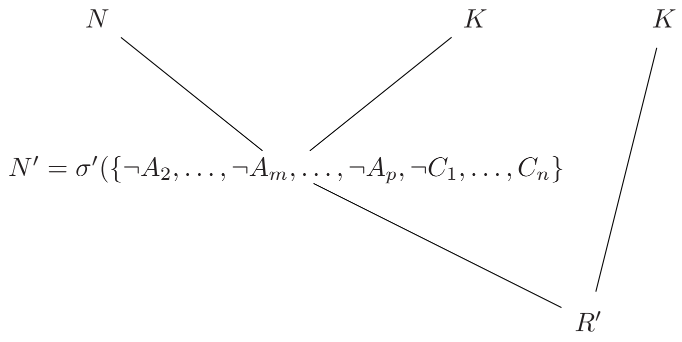
Sei $\mathcal{K}$ eine H-Klauselmenge. Dann ist $\mathcal{K}$ unerfüllbar $\Leftrightarrow$ $\Box$ ist aus einer negativen Klausel N in $\mathcal{K}$ durch binäre SLD-Resolution herleitbar.
Beweis:
Korrektheit: klar ✓
Vollständigkeit ("→"):
Wegen Vollständigkeit der vollen SLD-Resolution *(Satz 3.5.8)* existiert ein SLD-Resolutionsbeweis von $\Box$, der mit N startet.
Zeige: Jeder Resolutionsschritt mit voller SLD-Resolutions kann durch eine Folge binärer SLD-Resolutionsschritte ersetzt werden.
SLD Resolutionschritt:

$$N = \{\underline{\neg A_1}, \dots, \neg A_m, \dots, \neg A_p\} \qquad K = \{\underline{B}, \neg C_1, \dots, \neg C_n\}$$

$$\sigma = mgu(A_1, \dots, A_m, B)$$

$$R = \sigma(\{\neg A_{m+1}, \dots, \neg A_p, \neg C_1, \dots, \neg C_n\}$$

Durch Induktion über m zeigen wir, dass man diesen Schritt durch m binäre SLD-Resolutionsschritte ersetzen kann und den gleichen Resoventen R bis auf Variablenubenennung erhält.
IA: $m = 1$ trivial ✓ (dann ist das bereits binäre Resolution)
IS: m>1



$\overset{\Rightarrow}{Ind.Hyp.}$ geht auch in $m - 1$ binären Resolutionsschritten.
Es bleibt zu zeigen das $R = R'$ bis auch Variablenumbenennung. *qed*

16.05.2006

4 Logikprogramme

Zunächst: reine Logikprogramme
4.1: Syntax + Semantik
4.2: Universalität der LP
4.3: Indeterminismen der LP

4.1 Syntax und Semantik von Logikprogrammen

Bisher: Klausel = Menge von Literalen
Jetzt: Klausel = Folge von Literalen
Bisher: Klauselmenge = Menge von Klauseln
Jetzt: KLauslemenge = Folge von Klauseln
Also: Reihenfolge von Literalen / KLauseln spielt keien Rolle, Literate/Kl. können mehrfach auftreten

Definition 4.1.1 (Syntax von LP-Programmen)

Ein nicht-leere endliche Menge $\mathcal{P}$ von definiten Hornklauseln über der Signatur (Σ, Δ) heißt Logikprogramm über Signatur. Die Klauseln $\mathcal{P}$ heissen Programmklauslen. Man unterscheidet zwei Arten von Programm-Klauslen:

- Fakten sind Klauseln der Art $\{B\}$, B ist atomare Formel
- Regeln sind Klauseln der Art $\{B, \neg C_1, \ldots, \neg C_n\}$, $n \geq 1$, B,C_i at. Formel

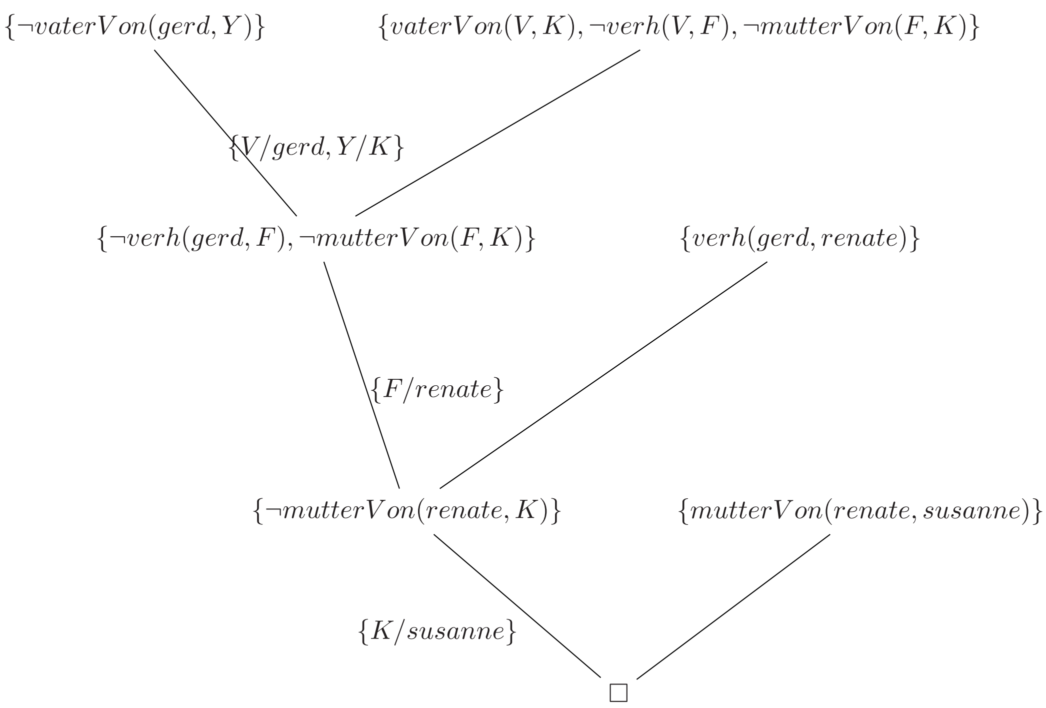
Aufruf eines LP geschieht durch eine

- Anfrage G der Art $\{\neg A_1, \ldots, \neg A_k\}$, $k \geq 1$

In der Prädik.Logik haben wir nur untersucht, ob eine Formel aus Formelmenge folgt. Bei LP will man auch noch wissen, wie die Variablen der Formel instanziiert werden müssen, damit die Formel aus der Formelmenge folgt $\Rightarrow$ „Antwortsubstution“
Bisher: Klausel entspricht der allq. Disjunktion ihrer Literale
Aufruf des LP P mit Anfrage G=$\{\neg A_1, \ldots, \neg A_k\}$: bedeutet, dass man untersuchen will, ob folgendes gilt:
$\mathcal{D} \models \exists X_1, \ldots, X_p A_1 \wedge \ldots \wedge A_k$ (*)
Variablen in P: implizit allquantifiziert.
Variablen in G: implizit existenzquantifiziert.
(*) ist äquivalent zu: $\mathcal{P} \cup \{G\}$ unerfüllbar
$\Leftrightarrow$ es ex. endl. Menge von Grundinstanzen von $\mathcal{P} \cup \{G\}$, die unerfüllbar ist (*Satz 3.3.9(a)*). Diese muss auch eine Grundinstanz von G enthalten. Wegen der Vollst. der SLD-Resolution reicht es, jeweils genau eine Grundinstanz von G zu betrachten.
$\Leftrightarrow$ es ex. Grundterme $t_1, \ldots, t_p$, so dass $\mathcal{P} \cup \{G[X_1/t_1, \ldots, [X_p/t_p]\}$ unerfüllbar
$\Leftrightarrow \mathcal{P} \models A_1 \wedge \ldots \wedge A_k \underbrace{[X_1/t_1, \ldots, X_p/t_p]}_{Antwortsub.}$.
Falls (*) gilt, soll das LP auch eine Antwortsubs. berechnen. Antwortsubs, die $X_1, \ldots, X_p$ durch Terme mit Variablen erstetzen. Die verbleibenden Variablen können dann weiter durch beliebige Terme ersetzt werden.

Beispiel 4.1.2

LP P:
mutterVon(reate,susanne).
verheiratet(gerd,renate).
vaterVon(V,K):- verheiratet(V,F), mutterVon(F,K).
Schreibweise von P als Klauselmenge $\mathcal{P}$ =
{{mutterVon(renate,susanne)}, {verheiratet(gerd,renate)}, {vaterVon(V,K),¬verheiratet(V,F),¬ mutterVon(F,K)}}
Anfrage: ?- vaterVon(gerd,Y).
D.h: $G = \{\neg$vaterVon(gerd,Y)$\}$
Satt $\mathcal{P} \models \exists Y$ vaterVon(gerd,Y) zu untersuchen, untersuche $\mathcal{P} \cup \{G\}$ durch binäre SLD-Resolution.



<u>Antwortsubstitution</u>

- Koposition der mgu's:
  - $\{K/susanne\} \circ \{F/renate\} \circ$ $\{V/gerd, Y/K\}$=$\{V/gerd, Y/susanne, F/renate, K/susanne\}$
- Einschränkung auf Variablen Y der Anfrage

3 verschiedene Arten zur Def. der Semantik von LP:

- deklarative Semantik
- prozedurale Semantik
- Fixpunkt Semantik

Diese sind alle äquivalent.

4.1.1 Deklarative Semantik der LP

Def. die Semantik für ein LP $\mathcal{P}$ zusammen mit Anfrage G.
Deklarative Sematik von $\mathcal{P}$ und G = alle „wahren“ Grundinstanzen von G, d.h. alle Grundinstanzen von G die aus $\mathcal{P}$ folgen.

Definition 4.1.3 (Deklarative Semantik)

Sei $\mathcal{P}$ ein LP und $G = \{A_1, \dots, A_k\}$ eine Anfrage. Dann ist die deklarative Semantik von $\mathcal{P}$ bezüglich G definiert als:
$D[\![\mathcal{P}, G]\!] = \{\sigma(A_1, \dots, A_k) | \mathcal{P} \models \sigma(A_1 \wedge \dots \wedge A_k),\ \sigma \text{ Grundsubst.}\}$
Jede Grundsubs $\sigma \in D[\![\mathcal{P}, G]\!]$ enthält als „Lösung“ die entspricht Instanz der Variablen aus $A_1, \dots, A_k$

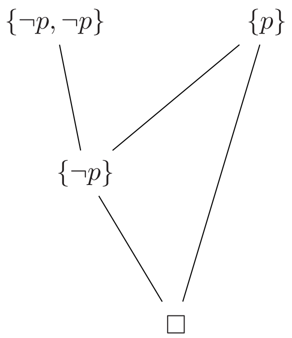
Beispiel 4.1.4

$\mathcal{P} \models \sigma(\text{vaterVon(gerd,Y)})$ gilt gdw. $\sigma(Y) =$susanne
$D[\![\mathcal{P}, G]\!] = \{\text{vaterVon(gerd,susanne)}\}$
$D[\![\mathcal{P} \cup \{\text{mutterVon(renate,peter}\}, G]\!] = \{\text{vaterVon(gerd,susanne),vaterVon(gerd,peter)}\}$.

4.1.2 Prozedurale Semantik der LP

Operationelle Semantik durch Angabe eines Interpreters. Interpreter arbeitet auf Konfigurationen: Paar aus Anfrage und Substitution.
Startkonfiguration: $(G, \emptyset)$ ($\emptyset$ = indentische Substitution) gewünschte Zielkonfiguration: $(\square, \sigma)$.
Antwortsubstitution ist dann σ eingeschränkt auf die Variablen von G. Man kann eine Konfiguration in die nächste durch binäre SLD-Resolution umformen. Zwei Unterschiede zur binären SLD-Resolution in Kapitel 3:

- Klauseln sind jetzt Folgen (statt Mengen) von Literalen
  z.B:



  Korr.+Vollst. bleibt erhalten.

- Statt Variablenumbenennungen auf beide Elternklauseln anzuwenden, darf man sie jetzt nur noch auf die definiten Programmklauseln anweden, nicht auf die jeweils andere negative Elternklausel. „standartisierte SLD-Resolution“

Definition 4.1.5 (Prozedurale Semantik eines LP)

Sei $\mathcal{P}$ ein LP

- Eine Konfiguration ist ein Paar (G, σ) aus eine Anfrage G und einer Subs. σ
- Es gibt ein Rechenschritt $(G_1, \sigma_1) \vdash_P (G_1, \sigma_1)$ gdw.
  - $G_1 = \{\neg A_1, \dots, \neg A_k\}$ mit $k \geq 1$

- es ex. Progklausel $K \in \mathcal{P}$ und eine Var-umb. ϑ mit $\vartheta(K) = \{B, \neg C_1, \dots, \neg C_n\}$ $n \geq 0$, so dass:
  - G_1 und $\vartheta(K)$ keine gemeinsammen Var. haben und
  - es ein $1 \leq i \leq k$ gibt, so dass A_i und B mit einem mgu σ unifizieraber sind
- $G_2 = \sigma(\{\neg A_1, \dots, \neg A_{i-1}, \neg C_1, ..., \neg C_n, \neg A_{i+1}, ... \neg A_k\})$
- $\sigma_2 = \sigma \circ \sigma_1$

- Eine Berechnung von $\mathcal{P}$ bei Eingabe von $G = \{\neg A_1, \dots, \neg A_n\}$ ist eine (endl. oder unendl.) Folge von Konfigurationen der Form $(G, \emptyset) \vdash_P (G_1, \sigma_1) \vdash_P (G_2, \sigma_2) \vdash_P ....$

- Eine Berechnung, die mit $(G, \emptyset)$ startet, heißt erfolgreich, falls sie mit $(\Box, \sigma)$ endet. Die Antwortsubs ist σ eingeschränkt auf die Var. aus G

- Prozedurale Semantik von $\mathcal{P}$ bezüglich G ist definiert als:
  $P[\![\mathcal{P}, G]\!] = \{\sigma'(A_1 \wedge \dots \wedge A_k) | (G, \emptyset) \vdash_P^+ (\Box, \sigma),\ \sigma'(A_1 \wedge \dots \wedge A_k)$ ist Grundinstanz von $\sigma(A_1 \wedge \dots \wedge A_k)\}$
  Hierbei ist „$\vdash_P^+$“ die transitive Hülle von „$\vdash_P$“
  (Es gilt $(G, \emptyset) \vdash_P^+ (\Box, \sigma)$ gdw $(G, \emptyset) \vdash_P \dots \vdash_P (\Box, \sigma)$).
  Analog dazu definieren wir „$\vdash_P^l$“ für $n \in \mathbb{N} : (G, \sigma) \vdash_P^- (G_l, \sigma_l)$ falls
  $(G, \sigma) \underbrace{\vdash_P (G_1, \sigma_1) \vdash_P \dots \vdash_P}_{l-Schritte} (G_l, \sigma_l)$ (l-Schritte)

19.5.2006

Beispiel 4.1.6

$\mathcal{P}$,G wie in Beispiel 4.1.2
$(\{\neg vaterVon(gerd, Y)\}, \emptyset) \vdash_P (\{\neg verheiratet(gerd, F), \neg mutterVon(F, K)\}, \{Y/K, V/gerd\})$
$\vdash_P (\{\neg mutterVon(renate, K)\}, \underbrace{\{F/renate\} \circ \{Y/K, V/gerd\}}_{\{F/renate, Y/K, V/gerd\}}$
$\vdash_P (\Box, \underbrace{\{K/susanne\} \circ \{F/renate, Y/K, V/gerd\}}_{\{K/susanne, F/renate, Y/susanne, V/gerd\}}$

Da die ursprüngliche Anfrage nur die Variable Y enthält:
Antwortsubstitution ist $\{Y/susanne\}$.
$P[\![\mathcal{P}, G]\!] = \{vaterVon(gerd, susanne)\}$

$\vdash_P$ hat zwei Indeterminismen:

- Wahl der Programmklausel K
- Wahl des Literals A_i

Beispiel 4.1.7

$\mathcal{P} = \{\{p(X, Z), \neg q(X, Y), \neg p(Y, Z)\}, \{p(U, U)\}, \{q(a, b)\}\}$
Anfrage G=$\{\neg p(V, b)\}$

$(\{\neg p(V, b)\}, \emptyset) \vdash_P (\{\neg q(V, Y), \neg p(Y, b)\}, \{X/V, Z/b\})$
$\vdash_P (\{\neg p(b, b)\}, \{V/a, Y/b, X/a, Z/b\})$
$\vdash_P (\neg q(B, Y'), \neg q(Y', b)\}, \{X'/b, Z'/b, V/a, Y/b, X/a, Z/b\})$
$\vdash_P (\{\neg q(b, b)\}, \{U/b, Y'/b, X'/b, Z'/b, V/a, Y/b, X/a, Z/b\})$

Die Herleitung ist endlich, aber nicht erfolgreich.
Alternativ:

$(\{\neg p(V,b)\},\emptyset) \vdash_P (\{\neg q(V,Y),\neg p(Y,b)\},\{X/V,Z/b\})$
$\vdash_P (\{\neg p(b,b)\},\{V/a,Y/b,X/a,Z/b\})$
$\vdash_P (\square,\{U/b,V/a,Y/b,X/a,Z/b\})$

Herleitung erfolgreich. Antwortsubsitution: $\{V/a\}$
Andere Alternative:

$(\{\neg(V,b)\},\emptyset) \vdash_P (\square,\{V/b\})$

Herleitung erfolgreich. Antwortsubstitution: $\{V/b\}$

Inderterminismen beeinflussen:

- Erfolg/Scheitern nach endlich vielen Schritten / unendliche Herleitung
- Unterschiedliche Antwortsubstitutionen bei erfolgreichen Pfad.

Satz 4.1.8 (Äquivalenz der deklarativen und prozeduralen Semantik (Clark))

Sei $\mathcal{P}$ ein LP und $G = \{\neg A_1,\ldots,\neg A_k\}$ eine Anfrage. Dann gilt $D[\![\mathcal{P},G]\!] = P[\![\mathcal{P},G]\!]$

Im Prinzip:
$\supseteq\triangleq$ Korrektheit der binären SLD-Resolution
$\subseteq\triangleq$ Vollständigkeit der binären SLD-Resolution
Aber: Die Antwortsubstitution muss noch mit berücksichtigt werden.

<u>Beweis:</u>

- Zeige erst $P[\![\mathcal{P},G]\!] \subseteq D[\![\mathcal{P},G]\!]$
  Sei $\sigma'(A_1 \wedge \ldots \wedge A_k) \in P[\![\mathcal{P},G]\!]$
  $\Rightarrow \underbrace{(G,\emptyset) \vdash_P \ldots \vdash_P (\square,\sigma)}_{l-Schritte}$, wobei $\sigma'(A_1 \wedge \ldots \wedge A_k)$ Grundinstanz von $\sigma(A_1 \wedge \ldots \wedge A_k)$ ist.
  <u>Induktion über l:</u>
  $(G,\emptyset) \vdash_P (G_1,\delta_1) \vdash_P (G_2,\delta_2\circ\delta_1) \vdash_P \ldots \vdash_P (\square,\delta_l\circ\ldots\delta_1)$
  Es existiert also ein A_i, ein $K \in \mathcal{P}$ mit $\nu(K) = \{B,\neg C_1,\ldots,\neg C_n\}$
  $\delta_1 = mgu(A_1,B)$, wobei $G_1 = \delta_1(\{\neg A_1,\ldots,\neg A_{i-1},\neg C_1,\ldots,\neg C_n,\neg A_{i+1},\ldots,\neg A_k\})$
  <u>IA:</u> $l = 1 \Rightarrow G_1 = \square$
  $\Rightarrow i = K = 1, n = 0$ (K ist Faktum)
  $\sigma = \delta_1$
  Zu zeigen ist: jede Grundinstanz von $\sigma(A_1)$ folgt aus $\mathcal{P}$
  Da $K \in \mathcal{P}, \nu(K) = \{B\}$ gilt $\mathcal{P} \models B$
  $\Rightarrow \mathcal{P} \models \underbrace{\delta_1(B)}_{=\delta_1(A)=\sigma(A)}$
  <u>IS:</u> l>1
  $(G_1,\emptyset) \vdash_P (G_2,\delta_2) \vdash_P \ldots \vdash_P (\square,\delta_l\circ\ldots\circ\delta_2)$ hat die Länge $l-1$
  <u>Induktions-Hypothese:</u> Alle Grundinstanzen der Atome in $\delta_l\circ\ldots\circ\delta_2(G_1)$ folgen aus $\mathcal{P} \Rightarrow$ Jede Grundinstanz von $\delta_l(\ldots\delta_2(\delta_1(\{A_1,\wedge\ldots\wedge A_{i-1}\wedge C_1\ldots\wedge C_n\wedge A_{i+1}\wedge\ldots\wedge A_k\}))\ldots)$ folgt aus $\mathcal{P}$
  Zu zeigen ist: Jede Grundinstanz von $\delta_l(\ldots\delta_2(\delta_1(\{A_1,\wedge\ldots\wedge A_k\}))\ldots)$ folgt aus $\mathcal{P}$. Ist klar für alle A_j mit $i \neq i$.
  Da alle Grundinstanzen von $\delta_l(\ldots\delta_2(\delta_1(\{C_1\ldots\wedge C_n\}))\ldots)$ aus $\mathcal{P}$ folgen, folgen auch alle Grundinstanzen von $\delta_l(\ldots\underbrace{\delta_1(B)}_{=\delta_1(A_i)},\ldots)$ aus $\mathcal{P}$. (Denn $\{B,\neg C_1,\ldots,\neg C_n\}$ ist Programmklausel).
- $D[\![\mathcal{P},G]\!] \subseteq P[\![\mathcal{P},G]\!]$. Verwende die Vollständigkeit der binären SLD-Resolution und eine Variante des Lifting-Lemmas.

4.1.3 Fixpunkt-Semantik der Logikprogammierung

Nicht modelltheoritisch definiert (wie die deklarative Semantik), sondern man versucht (ähnlich zur prozeduralen Semantik) durch resolutionsähnliche Schritte wahre Aussagen herzuleiten. Im Unterschied zur prozeduralen Semantik, gebe keine Anfrage vor, sondern gehe nur vom Programm aus und berechne alle Anfragen (ohne Variablen), die man in 0,1,2,3,... Schritten herleiten könnte:

Nach höchstens 0 Beweisschritten: keine Aussage ist beweisbar
Nach höchstens 1 Beweisschritten: alle Grundinstanzen von Fakten beweisbar (M_1) Nach höchstens 2 Beweisschritten: alle Aussagen, die durch höchstens **eine** Regelanwendung + Faktum beweisbar sind (M_2)

Die Funktion $trans_{\mathcal{P}}$:
$trans_{\mathcal{P}}(M)$ = alle Aussagen, die ausgehend von M mit höchstens einem weiteren Resolutionsschritt beweisbar sind.
M ist eine Menge von Aussagen (atomare Formeln ohne Variablen)
$M_0 = \emptyset$
$M_1 = tans_{\mathcal{P}}(M_0)$
$M_2 = trans_{\mathcal{P}}$
$M_i = tarns^i_{\mathcal{P}}(\emptyset)$

Definition 4.1.9 ($trans_{\mathcal{P}}$)

Sei $\mathcal{P}$ ein LP über (Σ, Δ). Dann definieren wir
$trans_{\mathcal{P}} : Pot(At(\Sigma, \Delta, \emptyset)) \mapsto Pot(At(\Sigma, \Delta, \emptyset))$ mit
$trans_{\mathcal{P}}(M) = M \cup \{A' \mid \{A', \neg B'_1, \ldots, B'_n\}$ ist Grunstinstanz von
$\{A, \neg B_1, \ldots, \neg B_n\} \in \mathcal{P}, B'_1, \ldots, B'_n \in M\}$

Beispiel 4.1.10

Sei $\mathcal{P}$ wie in Beispiel 4.1.2
$trans_{\mathcal{P}}(\emptyset) = \{mutterVon(renate, susanne), verheiratet(gerd, renate)\}$
$trans^2_{\mathcal{P}}(\emptyset) =$
$\{mutterVon(renate, susanne), verheiratet(gerd, renate)\} \cup \{vaterVon(gerd, susanne)\}$
$trans^3_{\mathcal{P}}(\emptyset) = trans^2_{\mathcal{P}}(\emptyset)$

23.05.06

- Deklarative + Prozedurale Semantik
- Fixpunkt-Semantik: $M_{\mathcal{P}} = \emptyset \cup trans_{\mathcal{P}} \cup trans^2_{\mathcal{P}} \ldots = \bigcup_{i \in \mathbb{N}} trans^i_{\mathcal{P}}(\emptyset)$ alle Aussagen die man in höchstens i Schritten herleiten kann.

Beispiel 4.1.11

LP:
$p(a).$
$p(f(X)) : -p(X).$
$trans_{\mathcal{P}}(\emptyset) = \{p(a)\}$
$trans^2_{\mathcal{P}}(\emptyset) = \{p(a), p(f(a))\}$
$trans^i_{\mathcal{P}}(\emptyset) = \{p(a), p(f(a)), \ldots, p(f^i(a))\}$
$M_{\mathcal{P}} = \{p(f^i(a) | i \geq 0\}$
$M_{\mathcal{P}}$ ist ein Fixpunkt von $trans_{\mathcal{P}} : trans_{\mathcal{P}}(M_{\mathcal{P}}) = M_{\mathcal{P}}$
Es ist sogar der **kleinste** Fixpunkt (bezüglich „$\subseteq$“).
$At(\Sigma, \Delta, \emptyset)$ ist immer Fixpunkt. Aber wir definieren die Semantik als <u>kleinsten</u> Fixpunkt, da er nur die Formeln enthalten soll, die man wirklich aus der leeren Menge herleiten kann.

Frage: Führt $M_{\mathcal{P}}$ immer zum kleinsten Fixpunkt?
Antwort: Ja!→ $trans_{\mathcal{P}}$ ist eine „stetige“ Funktion auf einer „vollständigen“ Ordnung.

Ordnung: transitive und antisymmetrische Relation.
„⊆“ ist eine **reflexive Ordnung**.

Lemma 4.1.12 (Vollständigkeit der Teilmengenrelation)

Die Relation ⊆ ist **vollständig**,d.h.

- es existiert ein kleinstes Element bezüglich ⊆
- für jede Kette $M_0 \subseteq M_1 \subseteq M_2 \subseteq \ldots$ existiert eine kleinste obere Schranke (least upper bound,lub). $M = \bigcup_{i\geq 0} M_i$

Beweis:

- Das kleinste Element ist $\emptyset$.
- M ist obere Schranke: $M_i \subseteq M = \bigcup_{i\geq 0} M_i$ ✓ M ist kleinste obere Schranke: Sei M' weitere obere Schranke von $M_0, M_1, \ldots$. $\Rightarrow M_i \subseteq M'$ f.a. i $\Rightarrow \bigcup_{i\geq 0} M_i \subseteq M'$ *q.e.d.*

Vollständige Ordnugen bezeichnet man auch als **complete partial orders** (cpo).
Wir betrachten die Kette
$\emptyset \subseteq trans_{\mathcal{P}}(\emptyset) \subseteq trans^2_{\mathcal{P}}(\emptyset) \ldots$
$M_{\mathcal{P}} = \bigcup_{i\geq 0} trans^i_{\mathcal{P}}(\emptyset)$ ist die kleinste obere Schranke dieser Kette.

Lemma 4.1.13 (Monotonie und Stetigkeit)

(a) Die Funktion $trans_{\mathcal{P}}$ ist monton, d.h. falls $M_1 \subseteq M_2$ dann $trans_{\mathcal{P}}(M_1) \subseteq trans_{\mathcal{P}}(M_2)$.
(„Aus größeren Megen von Aussagen lassen sich auch mehr Aussagen herleiten“)

(b) Die Funktion $trans_{\mathcal{P}}$ ist stetig, d.h. (BILD1) für jede Kette $M_0 \subseteq M_1 \ldots$ gilt:
$trans_{\mathcal{P}}(\bigcup_{i\geq 0} M_i) = \bigcup_{i\geq 0} trans_{\mathcal{P}}(M_i)$

Beweis:
(a): Folgt sofort aus Definition von $trans_{\mathcal{P}}$
(b): $trans_{\mathcal{P}}(\bigcup_{i\geq 0} M_i) \supseteq trans_{\mathcal{P}}(M_i)$, denn $trans_{\mathcal{P}}(\bigcup_{i\geq 0} M_i) \supseteq trans_{\mathcal{P}}$ wegen Monotonie in (a):
$trans_{\mathcal{P}}(\bigcup_{i\geq 0} M_i) \subseteq trans_{\mathcal{P}}$.
Sei $A' \in trans_{\mathcal{P}}(\bigcup_{i\geq 0} M_i)$. Dann ist $\{A', \neg B'_1, \ldots, \neg B'_n\}$ Grundinstanz einer Klausel aus $\mathcal{P}$ und $B'_1, \ldots, B'_n \in \bigcup_{i\geq 0} M_i$.
Da $M_0 \subseteq M_2 \ldots \Rightarrow$ es exisiter M_j mit $B'_1, \ldots, B'_n \in M_j$. Dann
$A' \in trans_{\mathcal{P}}(M_j) \subseteq \bigcup_{i\geq 0} trans_{\mathcal{P}}(M_i)$ *q.e.d.*
Fixpuntsatz (Tarski/Kleene):
Jede stetige Funktion f auf einer vollständigen Ordnung hat einen kleinsten Fixpunkt. Diesen erhält man als lub (least upper bound) der Kette:
$kleinstes_e lement, f(kleinstes_e lement), f(f(kleinstes_e lement)), \ldots$
Hier die spezielle Formulierung des Fixpunktsatzes für stetige Funktion $trans_{\mathcal{P}}$ und für die vollständige Ordnung ⊆.

Satz 4.1.14 (Fixpunktsatz)

Für jedes LP $\mathcal{P}$ hat $trans_{\mathcal{P}}$ eine kleinsten Fixpunkt (least fixpoint,lfp). Es gilt:
lfp($trans_{\mathcal{P}}$)=$\bigcup_{i\geq 0} trans^i_{\mathcal{P}}(\emptyset)$

Beweis:

(1) Zeige, dass $\bigcup_{i\geq 0} trans^i_{\mathcal{P}}(\emptyset)$ Fixpunkt von $trans_{\mathcal{P}}$ ist.

$trans_{\mathcal{P}}(\bigcup_{i\geq 0} trans^i_{\mathcal{P}}(\emptyset))$
$=\bigcup_{i\geq 0} \underbrace{trans_{\mathcal{P}}(trans^i_{\mathcal{P}}(\emptyset))}_{trans^{i+1}_{\mathcal{P}}(\emptyset)}$ *(Lemma 4.1.13(b))*
$= \emptyset \cup \bigcup_{i\geq 0} trans^{i+1}_{\mathcal{P}}(\emptyset)$
$= \bigcup_{i\geq 0} trans^i_{\mathcal{P}}(\emptyset)$

(2) Zeige, dass $\bigcup_{i\geq 0} trans^i_{\mathcal{P}}(\emptyset) \subseteq M$ für jeden Fixpunkt von $trans_{\mathcal{P}}$.

Wir zeigen $trans^i_{\mathcal{P}} \subseteq M$ für alle i durch Induktion.
IA: i=0: $\emptyset \subseteq M$ ✓
IS: $i > 0$:
Ind.Hyp: $trans^{i-1}_{\mathcal{P}}(\emptyset) \subseteq M$
$\Rightarrow \underbrace{trans(trans^{i-1}_{\mathcal{P}}(\emptyset)}_{trans^i_{\mathcal{P}}} \subseteq \underbrace{trans_{\mathcal{P}}(M)}_{=M(Fixpunkt)}$

Definition 4.1.15 (Fixpunkt-Semantik)

Sei $\mathcal{P}$ ein LP und $G = \{\neg A_1, \ldots, \neg A_k\}$ eine Anfrage. Dann ist die Fixpunkt-Semantik von $\mathcal{P}$ bezüglich G definiert als:
$F[\![\mathcal{P}, G]\!] = \{\sigma(A_1 \wedge \ldots \wedge A_k) | \sigma(A_i) \in lfp(trans_{\mathcal{P}})\}$

Satz 4.1.16 (Äquivalenz der Fixpunkt-Semantik zu den anderen Semantiken)

Sei $\mathcal{P}$ ein LP und G eine Anfrage. Dann gilt.
$D[\![\mathcal{P}, G]\!] = P[\![\mathcal{P}, G]\!] = F[\![\mathcal{P}, G]\!]$

Beweis:

(1) $P[\![\mathcal{P}, G]\!] \subseteq F[\![\mathcal{P}, G]\!]$. Sei $\sigma'(A_1 \wedge \ldots \wedge A_k) \in P[\![\mathcal{P}, G]\!]$. Dann ex. $(G, \emptyset) \vdash_P \ldots \vdash_P (\Box, \sigma)$. und $\sigma'(A_1 \wedge \ldots \wedge A_k)$ ist Grundinstanz von $\sigma(A_1 \wedge \ldots \wedge A_k)$.
Zu zeigen ist: $\sigma'(A_i) \in lfp(trans_{\mathcal{P}}) = \bigcup_{i\geq 0} trans^i_{\mathcal{P}}(\emptyset)$.
Es reicht zu zeigen:
Für alle A_i ex ein $j \geq 0$, so dass $trans^i_{\mathcal{P}}(\emptyset)$ alle Grundinstanzen von $\sigma(A_i)$ enthält. (analog zu prozedural↔deklarativ (Übung))

(2) $F[\![\mathcal{P}, G]\!] \subset D[\![\mathcal{P}, G]\!]$. Sei $\sigma(A_1 \wedge \ldots \wedge A_k) \in F[\![\mathcal{P}, G]\!] \Rightarrow \sigma(A_i) \in lfp(trans_{\mathcal{P}})$ für alle i.
Zu zeigen $\mathcal{P} \models \sigma(A_i)$.
Zeige stattdessen:
Für alle $A' \in trans^j_{\mathcal{P}}(\emptyset)$ gilt $\mathcal{P} \models A'$ (Induktion über j).
IA: j=0. Klar. $trans^0_{\mathcal{P}}(\emptyset) = \emptyset$
IS: $j > 0$: $A' \in trans_{\mathcal{P}}(trans^{j-1}_{\mathcal{P}}) \Rightarrow A' \in trans^{j-1}_{\mathcal{P}}(\emptyset)$. Es existiert Grunstinstanz $\{A', \neg B'_1, \ldots, \neg B'_n\}$ einer Klausel aus $\mathcal{P}$ mit $B'_1, \ldots, B'_n \in trans^{j-1}_{\mathcal{P}}(\emptyset)$.
Ind.Hyp: $\mathcal{P} \models B'_1 \ldots B'_n \Rightarrow \mathcal{P} \models A'$.

VL 26.05.2006

4.2 Universalität der Logikprogrammierung

Logikprogrammierung ist eine vollwertige Programmiersprache, man kann jede berechenbar Funktion auch durch ein Logikprogramm berechen. (Die Logikprogrammierung ist Turing-Vollständig Einschränkung auf arithmetische Funktionen $f : \mathbb{N}^n \to \mathbb{N}$) Andere Datenstrukturen kann man durch Abbildungen in $\mathbb{N}$ codieren.

Was sind berechenbare Funktionen?

Turing : alles, was mit Turing-Maschinen berechenbar ist.

Church : alles, was mit dem λ-Kalkül berechenbar ist.

Kleene : alle μ-rekursiven Funktionen.

Diese drei Aussagen sind äquivalent.
Church'sche These: Dieses ist eine Menge der im intuitiven Sinne berechenbaren Funktionen.

Definition 4.2.1 (μ-rekursive Funktionen)

Die Klasse der μ-rekursiven Funktionen ist die kleinste Klasse arithmetischer Funktionen mit:

1. Für jedes $n \in \mathbb{N}$ ist die Funktion $null_n : \mathbb{N}^n \to \mathbb{N}$ mit $null_n(k_1, \ldots, k_n) = 0$ μ-rekursiv.

2. Die Nachfolgerfunktion $succ : \mathbb{N} \to \mathbb{N}$ mit $succ(k) = k + 1$ ist μ-rekursiv.

3. μ-rekursive Funktionen sind unter **Komposition** abgeschlossen:
Falls $f : \mathbb{N}^m \to \mathbb{N}$, $f_1 : \mathbb{N}^n \to \mathbb{N}$,...,$f_m : \mathbb{N}^n \to \mathbb{N}$ μ-rekursiv sind, dann ist auch $g : \mathbb{N}^n \to \mathbb{N}$ μ-rekursiv mit $g(k_1, \ldots, k_n) = f(f_1(k_1, \ldots, k_n), \ldots, f_m(k_1, \ldots, k_n))$

4. μ-rekursive Funktionen sind unter **primitiver Rekursion** abgeschlossen:
Falls $f : \mathbb{N}^n \to \mathbb{N}$, $g : \mathbb{N}^{n+2} \to \mathbb{N}$ μ-rekursiv sind, dann ist auch $h : \mathbb{N}^{n+1} \to \mathbb{N}$ μ-rekursiv mit: $h(k_1, \ldots, k_n, 0) = f(k_1, \ldots, k_n)$
$h(k_1, \ldots, k_n, k+1) = g(k_1, \ldots, k_n, h(k_1, \ldots, k_n, k))$

5. μ-rekursive Funktionen sind unter **Minimalisierung** abgeschlossen:
Falls $f : \mathbb{N}^{n+1} \to \mathbb{N}$ μ-rekursiv ist, dann ist auch $g : \mathbb{N}^n \to \mathbb{N}$ μ-rekursiv mit:
$g(k_1, \ldots, k_n) = k \Leftrightarrow f(k_1, \ldots, k_n, k) = 0$ und für alle $0 \leq k' < k$ ist $f(k_1, \ldots, k_n, k')$ definiert, aber $f(k_1, \ldots, k_m, k') > 0$ (wenn $f(k_1, \ldots, k_n, k) \neq 0$ für alle k, dann ist $g(k_1, \ldots, k_n)$ nicht definiert)

Funktionen mit 1-5: **primitiv rekursive** Funktionen
Es existieren berechenbar Funktionen, die nicht primitiv-rekursiv sind:

- alle partiellen berechenbare Funktionen
- es existieren auch totale berechenbar Funtionen die nicht primitiv-rekuriv sind (Ackermann-Funktion)

Beispiel 4.2.2

- Addition:
$plus : \mathbb{N}^2 \to \mathbb{N}$ ist primitiv-rekursiv:
$plus(x, 0) = proj_{1,1}(x) \leftarrow x$
$plus(x, y+1) = f(x, y, plus(x, y)) \leftarrow (x, y) + 1$
$f(x, y, z) = succ(proj_{3,3}(x, y, z))$

- Multiplikation:
$times(x, 0) = null_1(x) \leftarrow 0$
$times(x, y+1) = g(x, y, times(x, y))plus(x, times(x, y))$
$g(x, y, z) = plus(proj_{3,1}(x, y, z), proj_{3,3}(x, y, z))plus(x, z)$

- Predecessor:
$p(0) = null_0$
$p(x+1) = proj_{1,2}(x, p(x))$

- Subtraktion:
  $minus(x,y) = 0$ falls $x \leq y$ $minus(x,y) = x - y$ sonst
  $minus(x,0) = proj_{1,1}(x)$
  $minus(x,y+1) = h(x,y,minus(x,y)) \leftarrow p(minus(x,y))$
  $h(x,y,z) = p(proj_{3,3}(x,y,z))$

- Division:
  $div(0,0) = 0$
  $div(x+1,0)$ undefiniert
  $div(x,y) = \lceil x/y \rceil$ sonst
  $div(x,y) = z \Leftrightarrow i(x,y,z) = 0$ und für alle $0 \leq z' \leq z$ ist $i(x,y,z')$ definiert und $i(x,y,z') > 0$
  $i(x,y,z) = minus(proj_{3,1}(x,y,z), j(x,y,z)) \leftarrow x - y * z$
  $j(x,y,z) = times(proj_{3,2}(x,y,z), proj_{3,3}(x,y,z)) \leftarrow x * z$

Darstellung von $\mathbb{N}$ in Logikprogrammen? $\leftarrow$ Logikprogramm arbeitet auf **Termen**
Berechenung von Funktionen in Logikprogrammen? $\leftarrow$ Logikprogramm definiert nur **Relationen**

Definition 4.2.3 (Berechnung arithmetischer Funktionen durch Logikprogramme)

- Jede Zahl $k \in \mathbb{N}$ wird durch den Term $\underline{k} \in \mathcal{T}(\Sigma, \Delta)$ mit $\underline{k} = \underbrace{s(\ldots s(0)}_{k-Stück}\ldots)$ dargestellt, wobei:
  $0 \in \Sigma_0, s \in \Sigma_1$.
  $(\underline{0} = 0, \underline{1} = s(0), \ldots)$
- Ein Logikprogramm $\mathcal{P}$ über (Σ, Δ) **berechnet** $f : \mathbb{N}^n \to \mathbb{N} \Leftrightarrow$ es existiert ein Prädikatssymbol $\underline{f} \in \Delta_{n+1}$ gibt, so dass $f(k_1, \ldots, k_n) = k \Leftrightarrow \mathcal{P} \models \underline{f}(\underline{k_1}, \ldots, \underline{k_n}, X)$

Beispiel 4.2.4

$\underline{plus}(X, 0, X)$.
$\underline{plus}(X, s(Y), s(Z)) :- \underline{plus}(X, Y, Z)$.
$\underline{times}(X, 0, 0)$.
$\underline{times}(X, s(Y), Z) :- \underline{times}(X, Y, U), \underline{plus}(X, U, Z)$.

Satz 4.2.5 (Universalität der Logikprogrammierung)

Jede μ-rekursive Funktion ist durch ein Logikprogramm berechenbar.

Beweis:
Induktion über den Aufbau der Klasse der μ-rekursiven Funktionen.

1. $\underline{null}_n(X_1, \ldots, X_n, 0)$.
2. $\underline{succ}(X, s(X))$.
3. $\underline{proj}_{n,i}(X_1, \ldots, X_n, X_i)$.
4. Es existieren $\underline{f}, \underline{f}_1, \ldots, \underline{f}_m$ Klauseln. Ergänze Logikprogramm um
   $\underline{g}(X_1, \ldots, X_n, Z) :- \underline{f}_1(X_1, \ldots, X_n, Z_1), \ldots, \underline{f}_n(X_1, \ldots, X_n, Z_m), \underline{f}(Z_1, \ldots, Z_m, Z)$.
5. Es existieren $\underline{f}, \underline{g}$ Klauseln. Ergänze Logikprogramm um
   $\underline{h}(X_1, \ldots, X_n, 0, Z) :- \underline{f}(X_1, \ldots, X_n, Z)$.
   $\underline{h}(X_1, \ldots, X_n, s(X), Z) :- \underline{h}(X_1, \ldots, X_n, X, Y), \underline{g}(X_1, \ldots, X_n, X, Y, Z)$.
6. Es existieren $\underline{f}$ Klauslen. Ergänze Logikprogamm um
   $\underline{g}(X_1, \ldots, X_n, Z) :- \underline{f}(X_1, \ldots, X_n, Z, 0), \underline{f'}(X_1, \ldots, X_n, Z)$. wahr falls für alle $0 \leq Z' < Z$ $\underline{f}(X_1, \ldots, X_n, Z')$ definiert ist und $\underline{f}(X_1, \ldots, X_n, Z') > 0$
   $\underline{f'}(X_1, \ldots, X_n, 0)$.
   $\underline{f'}(X_1, \ldots, X_n, s(X)) :- \underline{f'}(X_1, \ldots, X_n, X), \underline{f}(X_1, \ldots, X_n, X, s(Y))$.

VL 30.05.2006

Beispiel 4.2.6

$\underline{proj}_{3,3}(X,Y,Z,Z).$
$\underline{succ}(X,s(X))$
$\underline{f}(X,Y,Z,Z_2):-\underline{proj}_{3,3,}(X,Y,Z,Z_1),\underline{succ}(Z_1,Z_2).$
$\underline{proj}_{1,1}(X,X).$
$\underline{plus}(X,0,Z):-\underline{proj}_{1,1}(X,Z)\ \underline{plus}(X,s(Y),Z_2):-\underline{plus}(X,Y,Z_1),\underline{f}(X,Y,Z_1,Z_2).$

$\underline{div}(X,Y,Z):-\underline{i}(X,Y,Z,0),\quad \underbrace{\underline{i}'(X,Y,Z)}_{wahr, falls alle Z'<Z}$
$\underline{i}'(X,Y,0).$
$\underline{i}'(X,Y,s(Z)):-\underline{i}'(X,Y,Z),\underline{i}'(X,Y,Z,s(U)).$

4.3 Indeterminismus + Auswertungsstrategien

Ausführung von LP wie in prozeduraler Semantik: Falls an das LP $\mathcal{P}$ die Anfrage G gestellt wird, versucht man, eine erfolgreiche Berechnung:
$(G,\emptyset)\vdash_{\mathcal{P}}^{+}(\Box,\sigma)$
Ausgabe: Antwortsubstitution σ eingeschränkt auf die Variablen aus G.
$\vdash_{\mathcal{P}}$ hat in jedem Schritt $(G_1,\sigma_1)\vdash_{\mathcal{P}}(G_2,\sigma_2)$ hat zwei Arten von Indeterminismen:

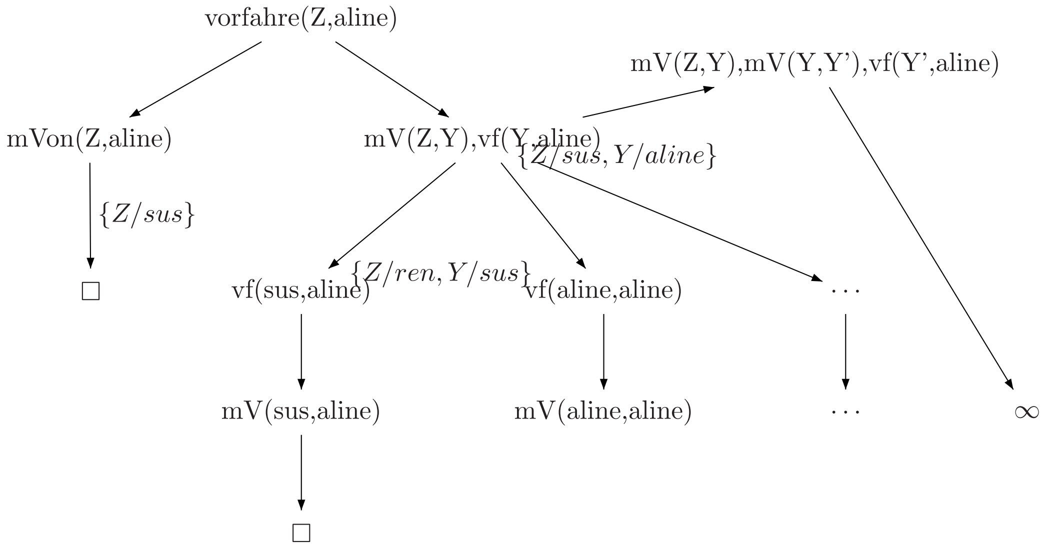
1. Wahl der Programmklausel, mit der resolviert wird (1.Art)

2. Wahl des Literals aus G_1, das zur Resolution verwendet wird (2.Art)

Beispiel 4.3.1

$mutterVon(reante, susanne)$
$mutterVon(susanne, aline)$
$vorfahre(V,X):-mutterVon(V,X).$
$vorfahre(V,X):-mutterVon(V,Y),vorfahre(Y,X).$

Anfrage: $?-vorfahre(Z,aline).$
$(\{\neg vorfahre(Z,aline)\},\emptyset)\vdash_{\mathcal{P}}(\{\neg mutterVon(Z,aline)\},\{V/Z,X/aline\})$
$(\{\neg vorfahre(Z,aline)\},\emptyset)\vdash_{\mathcal{P}}(\{\neg mutterVon(Z,Y),\neg vorfahre(Y,aline)\},\{V/Z,X/aline\})$
$\Rightarrow$ Indeterminismus 1.Art.



<u>Baum:</u> Statt $(\{\neg A_1, \ldots, \neg A_k\}, \Box)$ schreibe $A_1, \ldots, A_k$ an die Knoten. Markiere Knoten mit den in der Regel verwendeten mgu's eingeschränkt auf die Variablen der Anfrage.
<u>Baum hat:</u>

- erfolgreiche Pfade ($\Box$) (z.T. mit verschiedenen Antworten)
- endliche erfolglose Pfade (Widersprüche)
- unendliche Pfade

Auflösung der Indeterminismen beeinflusst, welcher Pfad im Baum zuerst betrachtet wird $\Rightarrow$ beeinflusst Programmverhalten.

Indeterminismus 2.Art ist „harmlos“:
Wenn man in auflöst (indem man nur Resolutionen mit dem 1.Literal einer Anfrage zulässt), dann entsteht der sogenannte **SLD-Baum**. Dieser Baum enthält immer noch jede erfolgreiche Berechnung mit jeder Antwortsubstitution, d.h.: Falls es eine Herleitung von $\Box$ mit Antwortsubstitution σ gibt, dann gibt es auch eine Herleitung von $\Box$ mit Anwortsubstitution σ' im SLD-Baum, σ und σ' unterscheiden sich nur durch Variablenumbenennung.
<u>Grund:</u> Bei Anfrage $G = \{\neg A_1, \ldots, \neg A_k\}$ kann man Resolutionsschritte, die erst mit $\neg A_i$ und dann mit $\neg A_j$ resolvieren, vertauschen.

Lemma 4.3.2 (Vertauschungslemma)

Seien $\{\neg A_1, \ldots, \neg A_k\}$, $\{B, \neg C_1, \ldots, \neg C_n\}$, $\{D, \neg E_1, \ldots, \neg E_m\}$ paarweise variablendisjunkt. Sei σ_1 mgu von A_i und B, sei σ_2 mgu von $\sigma_1(A_j)$ und D. Dann sind folgende SLD-Resolutionsschritte möglich:

$$\{\neg A_1, \ldots, \neg \underline{A_i}, \ldots, \neg A_i, \ldots, \neg A_k\} \qquad \{B, \neg C_1, \ldots, \neg C_n\}$$

$$\sigma_1(\{\neg A_1, \ldots, \neg C_1, \ldots, \neg C_n, \ldots, \neg \underline{A_j}, \ldots, \neg A_k\}) \qquad \{D, \neg E_1, \ldots, \neg E_m\}$$

$$\sigma_2(\sigma_1(\{\neg A_1, \ldots, \neg C_1, \ldots, \neg C_n, \ldots, \neg E_1, \ldots, \neg E_m, \ldots, \neg A_k\}))$$

Dann existiert auch ein mgu σ_1' von A_j und D und ein mgu σ_2' von $\sigma_1'(A_i)$ und B

$$\{\neg A_1, \ldots, \neg A_i, \ldots, \neg \underline{A_j}, \ldots, \neg A_k\} \qquad \{D, \neg E_1, \ldots, \neg E_m\}$$

$$\sigma_1'(\{\neg A_1, \ldots, \neg C_1, \ldots, \neg C_n, \ldots, \neg \underline{A_i}, \ldots, \neg A_k\}) \qquad \{B, \neg C_1, \ldots, \neg C_n\}$$

$$\sigma_2'(\sigma_1'(\{\neg A_1, \ldots, \neg C_1, \ldots, \neg C_n, \ldots, \neg E_1, \ldots, \neg E_m, \ldots, \neg A_k\}))$$

Dabei unterscheiden sich $\sigma_2 \circ \sigma_1$ und $\sigma_2' \circ \sigma_1'$ nur durch Variablenumbenennung d.h.:
$\nu \circ \sigma_2 \circ \sigma_1 = \sigma_2' \circ \sigma_1'$ für eine Variablenumbenennung ν

<u>Beweis:</u>
$\sigma_1(A_j)$ und $\underbrace{D}_{\sigma_1(D),\text{da Klauseln variablendisjunkt}}$ haben mgu σ_2.
d.h. $\sigma_2 \circ \sigma_1$ ist Unifikator von A_j und D. Dann haben A_j und D auch einen mgu $\sigma 1'$ und es exisitert Substitution σ mit: $\sigma_2 \circ \sigma_1 = \sigma \circ \sigma_1'$ (*)
Jetzt muss man zeigen, dass $\sigma_1'(A_i)$ und B unifizierbar sind. Gilt, denn σ ist ihr Unifikator:
$\sigma(\sigma_1'(A_i)) = \sigma_2(\underbrace{\sigma_1(A_i)}_{\sigma_1(B)})$ wegen (*)
$= \sigma_2(\sigma_1(B))$
$= \sigma(\underbrace{\sigma_1'(B)}_{\text{B, da Klauseln vardisj}})$
$= \sigma(B)$
Dann haben $\sigma_1'(A_i)$ und B auch einen mgu σ_2' und es existiert Substitution δ mit $\sigma = \delta \circ \sigma_2'$ (**)

Noch zu zeigen ist: $\sigma_2 \circ \sigma_1$ und $\sigma_2' \circ \sigma_1'$ sind bis auf Variablenumbenennung gleich.
Hierfür reicht es, zu zeigen: (a)$\sigma_2 \circ \sigma_1 = \delta \circ \sigma_2' \circ \sigma_1'$

(b)$\sigma_2' \circ \sigma_1' = \delta' \circ \sigma_2 \circ \sigma_1$
d.h. Substitutionen sind jeweils Instanzen voneinander. δ' kann man so erweitern, dass δ' eine Var-umb. wird (Übung).
Zeigen nun (a),(b) ist analog.
$\sigma_2 \circ \sigma_1 = \sigma \circ \sigma_1'$ (wegen (*))
$= \delta \circ \sigma_2' \circ \sigma_1'$ *q.e.d.*

VL 02.06.2006

Beispiel 4.3.3 (Illustration des Vertauschungslemmas)

```
p(Z,Z):- r(Z).\\
q(Z).
```

Anfrage: ?- p(X,Y),q(X).
$(\{\neg p(X,Y), \neg q(X)\}\emptyset) \vdash_{\mathcal{P}} (\{\neg r(Z), \neg q(Z)\}, \{X/Z, Y/Z\}) \vdash_{\mathcal{P}} (\{\neg r(Z)\}, \underbrace{\{W/Z}_{\sigma_2}, \underbrace{\{X/Z, Y/Z\}}_{\sigma_1})$

$(\{\neg p(X,Y), \neg q(X)\}\emptyset) \vdash_{\mathcal{P}} (\{\neg p(W,Y)\}, \{X/W\}) \vdash_{\mathcal{P}} (\{\neg r(Y)\}, \underbrace{\underbrace{\{W/Y, Z/Y}_{\sigma_2'}) \circ \underbrace{\{X/W\}}_{\sigma_1'}}_{\{W/Y, Z/Y, X/Y)}$

Ergebnis ist:
$\sigma_2 \circ \sigma_1(r(Z))$
$\sigma_2' \circ \sigma_1'(r(Z))$

Antwortsubstitutionen sind gleich bis auf Variablenumbenennung ν:
$\sigma_2' \circ \sigma_1'(r(Z)) = \nu \circ \sigma_2 \circ \sigma_1$ wobei $\nu = \{Y/Z, Z/Y\}$

Vertauschungslemma: Man kann beliebige Ordnungen der Literale wählen und bei der Resolution immer das ërsteLLiteral der Anfrage bearbeiten. $\Rightarrow$ Man kann beliebige Selektionsfunktionen (SLD) zur Auswahl des Literals verwenden. $\Rightarrow$ Betrachte Klauseln als Folgen von Literalen und verwende Selektionsfunktion, die das linkeste Literal der Anfrage auswählt.

Definition 4.3.4 (Kanonische Berechnung)

Eine Berechnung $(G_1, \sigma_1) \vdash_{\mathcal{P}} (G_2, \sigma_2) \vdash_{\mathcal{P}} \ldots$ heißt kanonisch, wenn in jedem Resolutionsschritt mit dem ersten Literal der jeweiligen Anfrage G_i resolviert wird.

Um zu zeigen, dass man sich auf kanonische Berechnungen einschränken kann, brauchen wir folgendes Lemma: Wenn sich zwei Anfragen nur durch Variablenumbenennung unterscheiden, dann kann man mit ihnen die gleichen Berechnugen durchführen und erhält die gleiche Antwortsubstitution (bis auf Variablenumbenennung).

Lemma 4.3.5

Sei $l \in \mathbb{N}$ Falls $(G, \sigma) \vdash_{\mathcal{P}}^{l} (G', \sigma')$ und ν eine Variablenumbenennung ist, dann $(\nu(G), \nu \circ \sigma) \vdash_{\mathcal{P}}^{l} (\nu(G'), \nu \circ \sigma')$

Beweis:
IA: $l = 0$ Trivial $(G' = G, \sigma = \sigma')$
IS: $l \geq 0$
$(G, \sigma) \vdash_{\mathcal{P}} (H, \delta \circ \sigma) \vdash_{\mathcal{P}}^{l-1} (G', \sigma')$
Ind.-Hyp.: $(\nu(H), \nu \circ \delta \circ \sigma) \vdash_{\mathcal{P}}^{l-1} (\nu(G'), \nu \circ \sigma)$
z.z.: $(\nu(G), \nu \circ \sigma) \vdash_{\mathcal{P}} (\nu(H), \nu \circ \delta \circ \sigma)$
Wir haben: $G = \{\neg A_1, \ldots, \neg A_k\}$

Es existiert eine variablenumbenannte Programmklausel (variablen-disjunkt mit G und $\nu(G)$) und $H = \delta(\{\neg A_1, \dots, \neg C_1, \dots, \neg C_n, \dots, \neg A_k\})$ mit $\delta = mgu(A_i, B)$.
Es genügt zu zeigen, dass $\nu \circ \delta \circ \nu^{-1} = mgu(A_i, B)$ (*)
Denn dann ergibt die Resolution von $\nu(G)$ und $\{B, \neg C_1, \dots, \neg C_n\}$:
$\nu(\delta(\nu^{-1}(\{\nu(A_1), \dots, \neg C_1, \dots, \neg C_n, \dots, \nu(A_k)\}))) = \nu(\underbrace{\delta(\{\neg A_1, \dots, \neg A_k\})}_{H}) = \nu(H)$

$\Rightarrow (\nu(G), \nu \circ \sigma) \vdash_{\mathcal{P}} (\nu(H), \nu \circ \delta \circ \nu^{-1} \circ \nu \circ \sigma)$

Beispiel 4.3.6 (Illustration von Lemma 4.3.5)

```
p(Z,Z):- r(Z).
q(W).
```

$(\{\neg p(X, Y), \neg q(X)\}, \sigma) \vdash_{\mathcal{P}} (\{\neg r(Y), \neg q(Y)\}, \{X/Y, Z/Y\} \circ \sigma)$
Sei $\nu = \{X/Y, Y/U, U/X\}$
$(\nu(\{\neg p(X, Y), \neg q(X)\}), \nu \circ \sigma) \vdash_{\mathcal{P}} (\{\neg r(U), \neg q(U)\}, \{Y/U, Z/U\} \circ \nu \circ \sigma)$

Satz 4.3.7 (Auflösung des Inderterminismus 2.Art)

Sei $\mathcal{P}$ ein LP und G eine Anfrage. Dann existiert zur jeder Berechnung $(G, \emptyset) \vdash_{\mathcal{P}}^{+} (\Box, \sigma)$ eine kanonische Berechnung $(G, \emptyset) \vdash_{\mathcal{P}}^{+} (\Box, \sigma')$ gleicher Länge, wobei sich σ und σ' nur durch Variablenumbenennung unterscheiden.

Beweis:
Wir haben eine Berechnung $(G, \emptyset) \vdash_{\mathcal{P}} (G_1, \delta_1) \vdash_{\mathcal{P}} (G_2, \delta_2 \circ \delta_1) \vdash_{\mathcal{P}} \dots \vdash_{\mathcal{P}} (\Box, \delta_l \circ \dots \circ \delta_1)$
Induktion über die Länge j der ersten nicht-kanonischen Teilberechnungsfolge.
IA: $j = 0 \Rightarrow$ die Berechnung ist bereits kanonisch.✓
IS: $j \geq 0$
Erster nicht-kanonischer Schritt sei $(G_i, \delta_i \circ \dots \circ \delta_1) \vdash_{\mathcal{P}} (G_{i+1}, \delta_{i+1} \circ \delta_i \circ \dots \circ \delta_1)$
Da man das erste Literal irgendwann durch Resolution eleminieren muss, um $\Box$ zu bekommen, muss später nocheinmal ein kanonischer Schritt kommen:
$i + j < l$
$(G_i, \delta_i \dots \circ \delta)$
nicht-kanonisch

$\vdash_{\mathcal{P}}^{j-1} (G_{i+j-1}, \delta_{i+j-1} \circ \dots \circ \delta_1)$ hier nochtmal genau gucken

Wende Vertauschnungslemma 4.3.2 an, um Schritte (*) und (**) zu vertauschen.
$\Rightarrow (G_{i+j-1}, \delta_{i+j-1} \circ \dots \circ \delta_1) \vdash_{\mathcal{P}}^{2} (\nu(G_{i+j-1}), \nu\delta_{i+j-1} \circ \dots \circ \delta_1)$ wobei der erste Berechnungsschritt kanonisch ist.
Nach Lemma 4.3.5: $(\nu(G_{i+j-1}), \nu \circ \delta_{i+j-1} \circ \dots \circ \delta_1) \vdash_{\mathcal{P}}^{l-i-j-1} (\Box, \nu\delta)$
D.h.:
kanonisch: $\vdash_{\mathcal{P}}^{i} (G_i, \delta_i \circ \dots \circ \delta_1)$
nicht-kanonisch: $\vdash_{\mathcal{P}}^{j-1} (G_{i+j-1}, \delta_{i+j-1} \circ \dots \circ \delta_1)$
nicht-kanonisch: $\vdash_{\mathcal{P}}^{2} (\nu(G_{+k+1}), \nu \circ \delta_{i+j+1} \circ \dots \circ \delta_1)$
$\vdash_{\mathcal{P}}^{l-i-j-1} (\Box, \nu \circ \delta)$

Lemma folgt jetzt aus der Induktion-Hypothese. Für den Fall j=1 ist eine extra Betrachtung notwendig. Zeige durch Induktion, dass man den nicht-kanonischen Schritt immer weiter nach hinten verschieben kann. Besser wäre eine Induktion über die Länge der erste kanonischen Teilberechnung.

Beispiel 4.3.8 (Einschränkung von Beispiel 4.3.1 auf kanonische Berechnugen)

Alle Lösungen werden immer nocht gefunden, aber unendliche Pfade fehlen.⇒ Der Indeterminismus der 2.Art beeinflusst das Terminierungsverhalten.
Der Baum der durch die Einschränkung auf kanonische Berechnungen entsteht, heißt SLD-Baum. Die Reihenfolge der Kinder entspricht der Reihenfolge der Programmklauslen.

VL 13.06.2006

Beispiel 4.3.9

```
p:- p.
q(a).
```

Anfrage: ?- q(b),p.

- Terminiert mit Fehlschlag wenn man mit linkem Literal beginnt ⇒ 2.Art beeinflusst Terminierungsverhalten, aber nicht die Vollständigkeit des Ableitungsbaums
- Terminiert nicht wenn man mit dem rechten Literal beginnt

Definition 4.3.10 (SLD-Baum)

Seien $\mathcal{P}$ ein LP, G eine Anfrage. Dann ist der **SLD-Baum** von $\mathcal{P}$ bei Anfrage G ein endlicher oder unendlicher Baum dessen Knoten mit Folgen von atomaren Formeln markiert sind. Der SLD-Baum ist der kleinste Baum mit folgenden Eigenschaften:

- Falls $G = \{\neg A_1, \ldots, \neg A_k\}$, dann ist die Wurzel mit $A_1, \ldots, A_k$ markiert
- Sei ein Konten mit $B_1, \ldots, B_n$ markiert, sei B_1 mit den positiven Literalen von k Programmklauseln $K_1, \ldots, K_k$ unifizierbar, wobei die Klauseln in dieser Reihenfolge im Programm sind. Dann hat Knoten k Nachfolger. Der i-te Nachfolger ist mit den Atomen markiert, die sich bei kanonischem Berechnungsschritt durch Resolution mit K_i ergeben. Falls Berechnung also die Gestalt $(\{\neg B_1, \ldots, \neg B_n\}, \emptyset) \vdash_{\mathcal{P}} (\{\neg C_1, \ldots, \neg C_n\}, \sigma)$ hat, so ist der i-te Nachfolgerknoten mit $C_1, \ldots, C_n$ markiert und die Kante mit σ eingeschränkt auf die Variablen $B_1, \ldots, B_n$ markiert.

Ablesen von Antwortsubstitutionen aus erfolgreichen Pfaden (enden mit $\Box$):
Wenn der Pfad mit $\delta_1, \delta_2, \ldots, \delta_l$ beschriftet ist, dann ist die Antwortsubstitution $\delta_l \circ \ldots \circ \delta_2 \circ \delta_1$ eingeschränkt auf die Variablen aus G.
Neben erfolgreichen Pfaden kann es auch noch:

- Pfade mit endlichen Fehlschlägen (enden mit mit Anfrage deren erstes Atom mit keiner Programmklausel resolvierbar ist)
- unendliche Pfade

SLD-Baum löst Indeterminismus 2.Art auf:

- repräsentiert Indeterminismus 1.Art: Reihenfolge der Kinder entsprechen Reihenfolge der Programmklauseln

Auflösung des Indeterminusmus 1.Art durch eine Auswertungstrategie, die angibt, wie der SLD-Baum zu durchsuchen/aufzubauen ist. Untersuche danach ob man an einer oder an allen Lösugen interessiert ist.

Breitensuche ist vollständig, d.h. jede erfolgreiche Berechnung wird irgendwann gefunden.
Nachteil: ineffizient

Tiefensuche ist unvollständig, d.h. nicht jede erfolgreiche Berechnung wird gefunden wenn es unendliche Pfade gibt.
Vorteil: kann effizienter sein

Prolog verwendet Tiefensuche, wobei linkester Pfad zuerst betrachtet wird. Bei Eingabe von “;“ findet Rücksetzen/Backtracking statt

⇒ Programmierer sollten Anordnung von Literalen in Klauslen und von Klauseln im Programm “geschickt“ wählen. Vertauschung von Literalen kann Effizient und Terminierung beeinflussen.

Beispiel 4.3.11 (Vertauschung der Literale in rekursiver VF-Klausel)

⇒ findet immer nocht beide Lösungen, aber wenn man “;“ ⇒ Nicht-Terminierung

Beispiel 4.3.12 (Vertauschung der letzten beiden Klauseln)

⇒ Erste Anfrag terminiert nicht.

- Nichtrekursive Klaueln eines Prädikates sollten vor den rekursiven Klauseln des Prädikates stehen
- Indeterminismus 1.Art: Auflösung in Prolog: bearbeite Programmklauseln von oben nach unten
- Indeterminismus 2.Art: Auflösung: bearbeite Literale von links nach rechts

5 Die Programmiersprach Prolog

Bekannteste Sprache, die auf LP beruht (1.Hälfte der 70er).
Popularität in der KI: Hauptsprache des japanischen “Fifth Gerenation Project“ (1981)

Syntax von (einfachem) Prolog: “:-“, “?-“
Signatur ergibt sich aus den auftretenden Funktions-/Prädikatssymbolen
(beginnen mit Kleinbuchstaben oder Strings aus Sonderzeichen `(<-->)`oder Strings in Apostrophen $'X'$)
Variablen beginnen mit Großbuchstaben oder `_`
Besonderheit: `_` anonyme Variable

- Mehrfache vorkommen dürfen unterschiedliche instantiiert werden.
- Belegungen von `_` werden bei Antwortsubstitution nicht ausgegeben

Beispiel: p(a,b,c).
Anfrage: ?-p($\_, \_, X$).
Antwort: X=c

Überladen von Funktions-/Prädikatssymbolen ist möglich

```
p(a,b,c).
p(a).
p(a,p(b,c)).
```

$p \underbrace{/3}_{Stelligkeit}$ und p/2 haben nichts miteinander zu tun

Semantik von (einfachem) Prolog:Semantik von LP, wobei der SLD-Baum in Tiefensuche von links nach rechts druchlaufen wird.
Aber: Keine korrekte Unifikation, sondern Unifikation **ohne occur-check** (aus Effizienzgründen)

Wenn X und t unifiziert werden sollen und $X \neq t$, dann überprüft Prolog nicht, ob X in t auftritt. Speicherzelle für X zeigt dann auf die Speicherzelle von t. X, t=f(X).
$\Rightarrow X = f(f(f(\ldots)))$ unendlicher Term. X und f(X) hat den mgu=$\{X/f(f(f(\ldots)))\}$ wenn man unendliche Terme zulässt.

Programm:

```
p(X,f(X)).
```

Anfrage: ?- p(X,X). Antwort: X=f(f(...))

Es existiert ein vordefiniertes Prädikat `unify_with_occur_check/2` :
?-$unify\_with\_occur\_check(X, f(X)) \Rightarrow$ No
?-$unify\_with\_occur\_check(X, f(Y)) \Rightarrow X = f(Z), Y = Z$

Folgende Abschnitte: Eigenschaften von Prolog, die über reine LP hinausgehen.

5.1 Arithmetik

LP arbeitet auf Termen $\Rightarrow$ Datenobjekte müssen als Terme dargestellt werden. $\mathbb{N}$: z.B. als Terme über $0 \in \Sigma_0$, $s \in \Sigma_1$: 0=0,s(0)=1,usw.

Addition

add(X,Y,Z): "X+Y=Z"
2-stellige Funktion wird durch 3-stelliges Prädikat ausgedrückt.
add(X,0,0). add(X,s(Y),s(Z)):- add(X,Y,Z).

Nachteil:

- Lesbarkeit
- Effizienz

$\Rightarrow$ Prolog biete Unterstützung für Zahlen und Listen

Vorteile:

- Bidirektionalität (Ein- und Ausgabe ist nicht festgelegt)

?-add(s(0),s(s(0)),X).$\Rightarrow$ X=s(s(s(0))), berechnet 1+2
?-add(X,s(s(0)),s(s(s(0)))).$\Rightarrow$ X=s(0), berechnet 3-2

VL 16.06.2006

```
?- add(X,Y,s(s(s(0)))). -> 4 L"osungen
?- add(X,s(s(0)),Z). -> X=U, Z=s(s(U))
?- add(s(0),Y,Z). -> unendlich viele Antwortsubstitutionen
```

$\Rightarrow$ Fast jedes Prolog Programm kann bei einer entsprechenden Anfrage zu einem unendlichen SLD-Baum führen.

- Eingebaute natürliche Zahlen:
  Arithmetischer Ausdruck: Term der aus Zahlen, Variablen, bestimmten vordefinierten binären Infix-Funktionen (+,-,*,//,**,...), Präfix-Funktion "-"

Die Funktionssymbole +,-,... kann man wie bisher mit syntaktischer Unifikation behandeln:

```
Prog: equal(X,X).
?- equal(3,1+2). -> No
?- equal(X,1+2). -> X=1+2
```

Auswertung von +,-,... geht nur durch spezielle vordefinierte Prädikate. Vordefinierte Prädikatssymbole zum **Vergleich** von arithmetischen Ausdrücken: $op \in \{$`<,>,=<,>=,=:=,=\=`$\}$
?-t_1 op t_2 ist erfolgreich, wenn t_1, t_2 voll instantiierte arithmetische Ausdrücke sind und wenn das Resultat z_1 der Auswertung von t_1 und z_2 die von t_2 in der Relation op stehen. Programm bricht mit Fehler ab, wenn t_1 oder t_2 kein voll instantiierte arithmetischer Ausdruck ist. $\Rightarrow$ op erzwingen Auswertung ihrer Argumente.

```
?- 1 < 2. -> Yes
?- -2 < -1. -> Yes
?- 1*1 < 1+1. -> Yes
?- 2 < 1. -> No
?- 6//3 < 5-4. -> No
?- a < 1. -> Programmabbruch
?- X < 1. -> Programmabbruch
?- X =:= 2. -> Programmabbruch (nicht X=2)
```

$\Rightarrow$ Weiteres vordefiniertes Prädikatssymbol "is"

?- t_1 is t_2.
ist erfolgreich, wenn t_2 ein voll instantiierte arithmetischer Ausdruck ist, der zu einem Wert z_2 auswertet und wenn t_1 mit z_2 unifiziert. Programmabbruch wenn t_2 kein voll instantiierte arithmetischer Ausdruck ist.

```
?- 2 is 1+1. -> Yes
?- 2 is 2. -> Yes
?- 1+1 is 2. -> No
?- X+1 is 1+1. -> No
?- X is 2. -> X=2
?- X is 1+1. -> X=2
?- X is 3+4, Y is X+1. -> X=7, Y=8
?- Y is X+1, X is 3+4. -> Programmabbruch
?- X is X., ?- 2 is X., ?- X is a. -> Programmabbruch
```

Weitere vorderfinierte Gleichheit "=" durch Faktum X=X. (reine syntaktische Unifiktion)

```
?- a = a. -> Yes
?- 2 = 2. -> Yes
?- 1+1 = 1+1. -> Yes
?- 2 = 1+1. -> No
?- X+1 = 1+1. -> X=1
?- X = 1+1. -> X=1+1
?- X = X. -> X=Y
?- 1+X = Y+1. -> X=1, Y=1
?- X = 3+4, Y is X+1. -> X=3+4, Y=8
?- X = f(X). -> X=f(f(...))
```

Gleicheit:
Termgleicheit
= (syntaktisch)
`unify_with_occurs_check`(korrekt)

Wertgleicheit
is
=:=

<u>“add“ mit vordefinierten Zahlen:</u>

```
add(X,0,X).
add(X,Y,Z):- Y>0, Y1 is Y-1, add(X,Y1,Z1), Z is Z1+1.
(alternativ: add(X,Y,Z):- Z is X+Y.
?- add(1,2,X). -> X=3
?- add(X,2,3). -> Programmabbruch
```

⇒ Bidirektionalität kann verloren gehen

```
add(X,0,X).
add(X,Y+1,Z+1):- add(X,Y,Z).
?- add(1,2,X). -> No
?- add(1,0+1,x). -> X=1+1
=> Nicht vorteilhaft
```

Beispiel (Fakultät, ggT)

```
fakt(0,1).
fakt(X,Y):- X>0, X1 is X-1, fakt(X1,Y1), Y is Y1*X.

ggT(0,X,X).
ggT(X,0,X).
ggT(X,Y,Z):- X=<Y, X>0, Y1 is Y-X, ggT(X,Y1,Z).
ggt(X,Y,Z):- X>>, Y>0, X1 is X-Y, ggT(X1,Y,Z).

?- fakt(3,X). -> X=6
?- ggT(28,36,X). -> X=4
```

<u>Weitere vordefinierte Prädikate:</u>

```
number/1
number(t) ist wahr, falls t eine Zahl ist
?- number(2). -> Yes
?- X is 1+1, number(X). -> Yes
?- number(1+1). -> No
?- number(X). -> No
```

5.2 Listen

Darstellung von Listen als Terme:
nil $\in \Sigma_0$ (leere Liste)
cons $\in \Sigma_2$ (Einfügen vorne in Liste)
(cons(1,cons(2,nil)) entspricht [1,2])

```
len(nil,0).
len(cons(X,XS),Y):- len(XS,Y1), Y is Y1+1
?- len(cons(7,cons(3,nil)),X). -> X=2
```

Prolog bietet eine lesbarere Kurzschreibweisen für Listen, falls man statt “nil“ das Funktionssymbol `[]` und statt “cons“ das Funktionssymbol `.` benutzt.

```
len([],0).
len(.(X,XS),Y):- len(XS,Y1), Y is Y1+1.
```

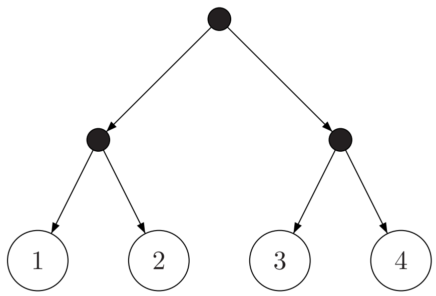
Folgende Kurzschreibweisen sind möglich:

- $.(t_1, t_2) = [t_1|t_2]$
- $.(t, []) = [t]$
- $.(t_1, .(t_2, .(t_3, t))) = [t_1, t_2, t_3|t]$
- $.(t_1, .(t_2, .(t_3, []))) = [t_1, t_2, t_3] = [t_1, t_2|[t_3|[]]] = [t_1|[t_2, t_3|[]]]$

Kurzschreibweisen werden als **identisch** zu Originalterm aufgefasst.

```
?- [1,2] = [1|[2]]. -> Yes
?- .(1,.(2,[])) = [1,2]. -> Yes
?- .(1,2) = [1|2]. -> Yes
?- .(1,X) = [1,2,3]. -> X=[2,3]
?- .(X,.(1,X)) = [[2],Y]. -> X=[2], Y=[1,2]
```

"." ist rein syntaktisches Funktionssymbol. Kann man auch für Binärbäume benutzten.




```
= .(.(1,2).(3,4))
= [[1|2],[3|4]]
```

VL 20.06.2006

Beispiel

```
member(X,[X|_).
member(X,[_|YS):- member(X,YS).

?- member(X,[[a,b],1,[]]).
X=[a,b];
X=1;
X=[];
No.

?-member(b,XS). Welche Listen enhalten b?
XS=[b|YS]; Alle Listen mit 1. Element b
XS=[Y,b|YS]; Alle Listen mit 2. Element b
...
```

```
app([],YS,YS).
app([X|XS],YS,[X|ZS]):- app(XS,YS,Z).

?- app([1,2],[3,4],X).
X=[1,2,3,4]

?- app(XS,YS,[1,2,3]).
XS=[], YS=[1,2,3];
XS=[1], YS=[2,3];
...
No.

?- app(XS,[],ZS).
XS=[], ZS=[];
XS=[X], ZS=[X];
...
unendliche viele Antworten
```

5.3 Operatoren

Bisher: Schreibe Terme und atomare Formeln in Präfix-Notation mit Klammer. `p(X,f(a))` $p \in \Delta_2, f \in \Sigma_1, a \in \Sigma_0$. p,f,a sind sogenannte **Funktoren**.
Jetzt: Prädikats-/Funktionssymbole in Infix-,Präfix-, Postfixschreibweise ohne Klammern (**Operatoren**). Grund: Bessere Lesbarkeit („Programmieren mit natürlicher Sprache“)

Beispiel

+,-,* sind bereits als Operatoren vordefiniert.
„2+3“ wird von Prolog in +(2,3) umgewandelt.

Benutzer kann selbst neue Operatoren deklarieren:
Direktive:

```
:- op(Präzedenz,Typ,Name(n)).
```

Direktive = Programmklauseln ohne Kopf (Anfragen). Beim Laden versucht Prolog diese Aussagen zu beweisen.
Neu deklarierte Operatoren sind erst nach der entsprechenden Direktive verwendbar.

Vordefinierte Direktiven

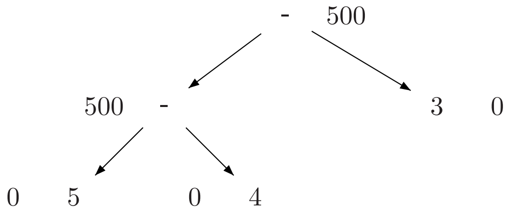
```
:- op(500,yfx,[+,-]).
:- op(400,yfx,*).
```

1. Argument: Zahl zwischen 0 und 1200 gibt an, wie stark der Operator bindet. Kleinere Präzedenz entspricht stärkere Bindung.
2. Argument: Bestimmt Reihenfolge von Operator und Argument(en)
f entspricht Operator, x,y entsprechen Argumenten.
xfy, yfx, xfx sind **binäre Infix-Operatoren**.
fx,fy sind **unäre Präfix-Operatoren**.
xf,yf sind **unäre Postfix-Operatoren**.
3. Argument: Name/Liste von Namen von Funktions- oder Prädikatssymbolen

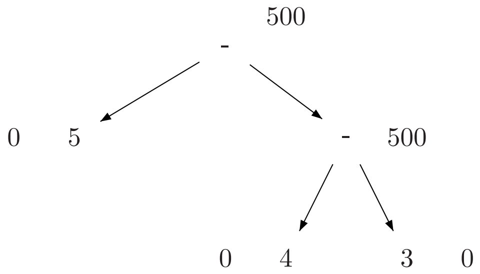
$$5-4-3=\begin{cases}(5-4)-3=-2\\5-(4-3)=4\end{cases}$$

x = Argumente mit einer Präzedenz **echt kleiner** als die Präzedenz von f ist.
y = Argumente mit einer Präzedenz **kleiner oder gleich** als die Präzedenz von f ist.

(5-4)-3:



5-(4-3):



Kann nicht sein, denn das rechte Argument von „-“ muss eine Präzedenz von < 500 haben.

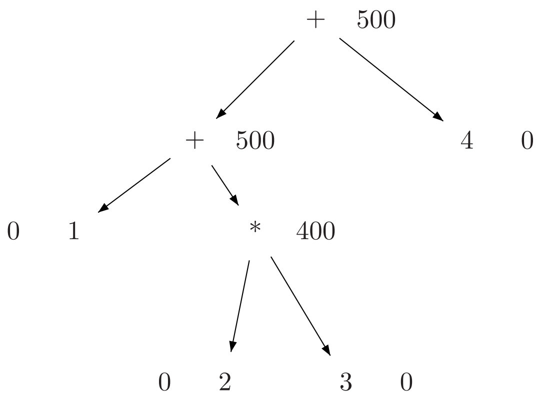
```
?- 5-4-3 =:= -2.
Yes.
```

„yfx“ bedeutet **Linksassoziativität**.

```
?- 1+2+3 = 1+(2+3).
No.

?- 1+2+3 = (1+2)+3.
Yes.
```

„xfy“ bedeutet **Rechtsassoziativität**, „xfx“ keine Assoziativität. Bei `:-op(500,xfx,+).` ist „1+2+3“ nicht möglich ⇒ Programmabbruch.

```
1+2*3+4 : 1+(2*3)+4
```



Operatoren dürfen auch überladen werden:
`:-op(200,fy,-).` -2-3: (-2)-3.

Definiere eingene Operatoren für einfache Form von Sprachverarbeitung:

- Verb „was“, 2-stellig, Infix-Schreibweise. „laura was young“ = was(laura,young). Keine Assoziativität („laura was young was beautiful“ ist sinnlos). **Typ:** xfx

- „of“, 2-stellig, Infix, „secretary of john“. Rechassoziativ „secretary of son of john“. **Typ:** xfy. „laura was secretary of john“. Niedrigere Präzedenz als „was“

- „the“, 1-stellig, Präfix, keine Assoziativität. **Typ:** fx. „the secretary of the son“. Niedrigere Präzedenz als „of“.

Programm

```
:-op(300,xfx,was).
:-op(250,xfy,of).
:-op(200,fx,the).

laura was the secreaty of the head of the department.

?- Who was the secretary of the head of the department.
Who = laura.

?- laura was Who.
Who = the secretary of the head of the department
```

5.4 Das Cut-Prädikat und Negation

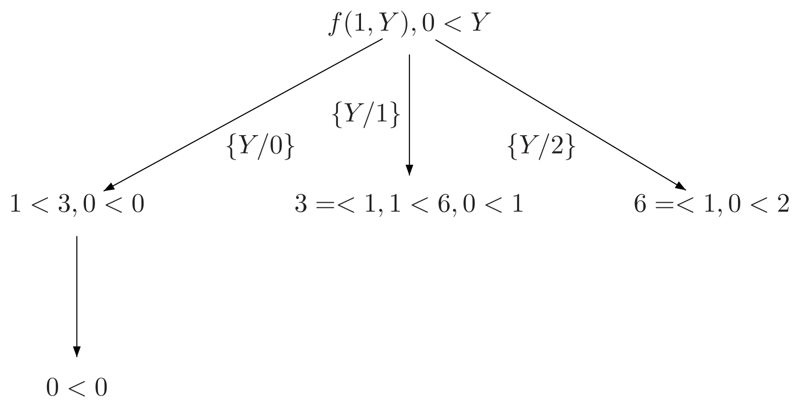
Prolog führt automatisch Backtracking durch, wenn es einen endlichen Fehlschlag erkenn (Blatt$\neq \square$). Kann nachteilig sein, da dies sowohl zeit- als auch speicherintensiv ist (frühere Knoten mit Alternativen müssen gespeichert werden) und auch zur Nichterminierung führen kann. $\Rightarrow$ Beschneide SLD-Baum so, dass bestimmte Kanten durch Zurücksetzen nicht druchlaufen werden dürfen.

VL 23.06.2006

Beispiel 5.4.1

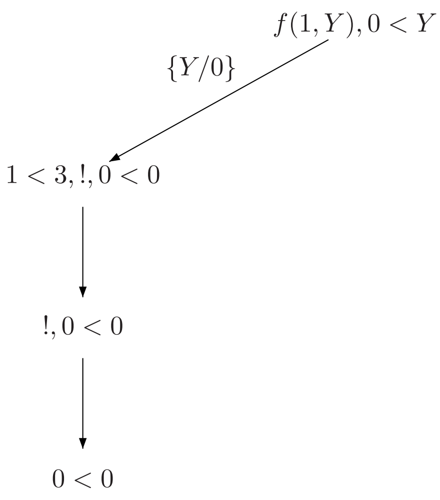
$$f(x) = \begin{cases} 0 & x < 3 \\ 1 & 3 \leq x < 6 \\ 2 & x \geq 6 \end{cases}$$

```
f(X,0):- X<3.
f(X,1):- 3=<X, X<6.
f(X,2):- 6=<X.

?- f(1,Y),0<Y.
```



<u>Ergebnis:</u> No. Das Ergebnis kann erst ausgegeben werden, wenn der gesammte SLD-Baum aufgebaut wurde. Dies kann sehr ineffizient sein (bzw. nicht-terminierend).
<u>Ziel:</u> Verbessere Programm: Untersuche, welche Bedingungen im Programm sich ausschließen$\Rightarrow$ falls manche Teilziele bewiesen werden konnten, muss man manche anderen Teilziele gar nicht erst untersuchen.
$\Rightarrow$ Falls $X < 3$ zutrifft, dann braucht man die zweite Klausel nicht mehr betrachten, und man braucht die dritte Klausel auch nicht mehr zu betrachten.
$\Rightarrow$ Falls $X < 6$ zutrifft, dann braucht man die dritte Klausel nicht mehr zu betrachten. Im Beispiel: Da „$1 < 3$" bewiesen wurde, sollte man den mittleren und den rechten Pfad des SLD-Baums nicht aufbauen.
In Prolog gibt es dazu das 0-stellige Prädikatssymbol **Cut**: „**!**". Diese darf in rechten Seiten von Regeln und in Anfragen auftreten. Es ist immer beweisbar, aber es schneidet Pfade im SLD-Baum ab.

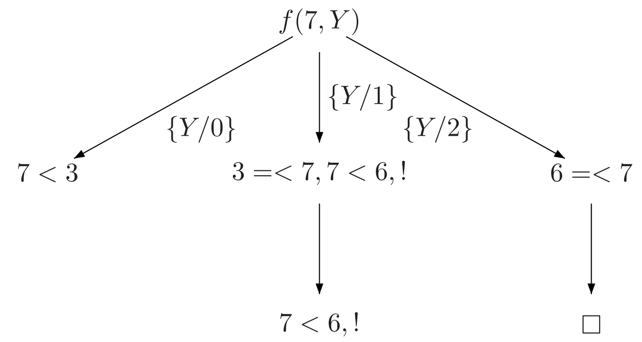
`f(X,0):-X<3,!.` (Falls Beweis von $X < 3$ gelingt, dann wird druch ! das Rücksetzen verhindert $\Rightarrow$ keine Betrachtung der anderen f-Klauseln).
`f(X,1):-3=<X,X<6!.`
`f(X,2):-6=<X.`



Nach dem Gelingen von $X < 3$ wird mit keiner anderen f-Klausel beim Rücksetzten resolviert.
„grüne Cuts": beeinflussen die Effizienz, aber nicht das Ergebnis.
„rote Cutes": Weglassen der Cuts ändert nicht nur die Effizienz, sondern auch das Ergebnis.

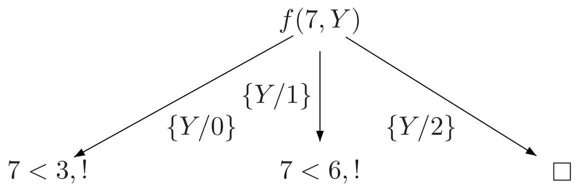
Beispiel 5.4.2

`?-f(7,Y).`



Antwort: Y=2.
Ineffizient: Falls $X < 3$ scheitert, muss $3 =< X$ gelingen. Falls $X < 6$ scheitert, muss $6 =< X$ gelingen.

```
f(X,0):- X<3,!.
f(X,1):- X<6,!.
f(X,2).
```



`?-f(1,Y).` Antwort: Y=0;$\underbrace{Y = 1; Y = 2}_{\text{ohne Cuts}}$

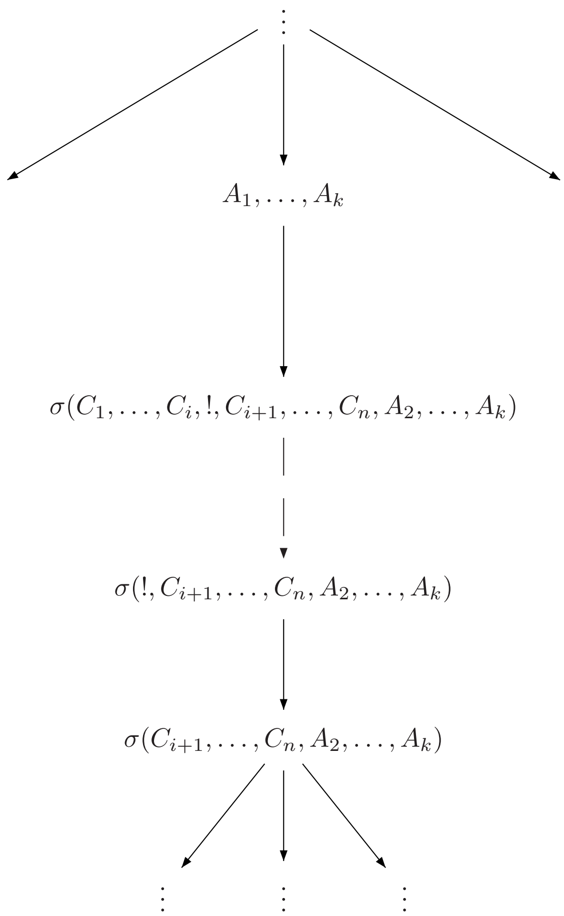
Bei Verwendung von Cuts denkt man an eine „bestimmte Verwendung“ des Prädikats (bestimmte Positionen bei Ein- und Ausgabe). Im Beispiel: 1.Argument entspricht Eingabe, 2.Argument entspricht Ausgabe.
Bei anderer Verwendung können Cuts zu „unerwarteten“ Ergebnissen führen.
`?-f(0,Y).` $\Rightarrow Y = 0$
`?-f(0,2).` $\Rightarrow$ `Yes.`

Beispiel 5.4.3

```
p(0):-!.
p(1):-!.

?-p(X). Antwort: X=0.
?-p(1). Antwort: Yes.
```

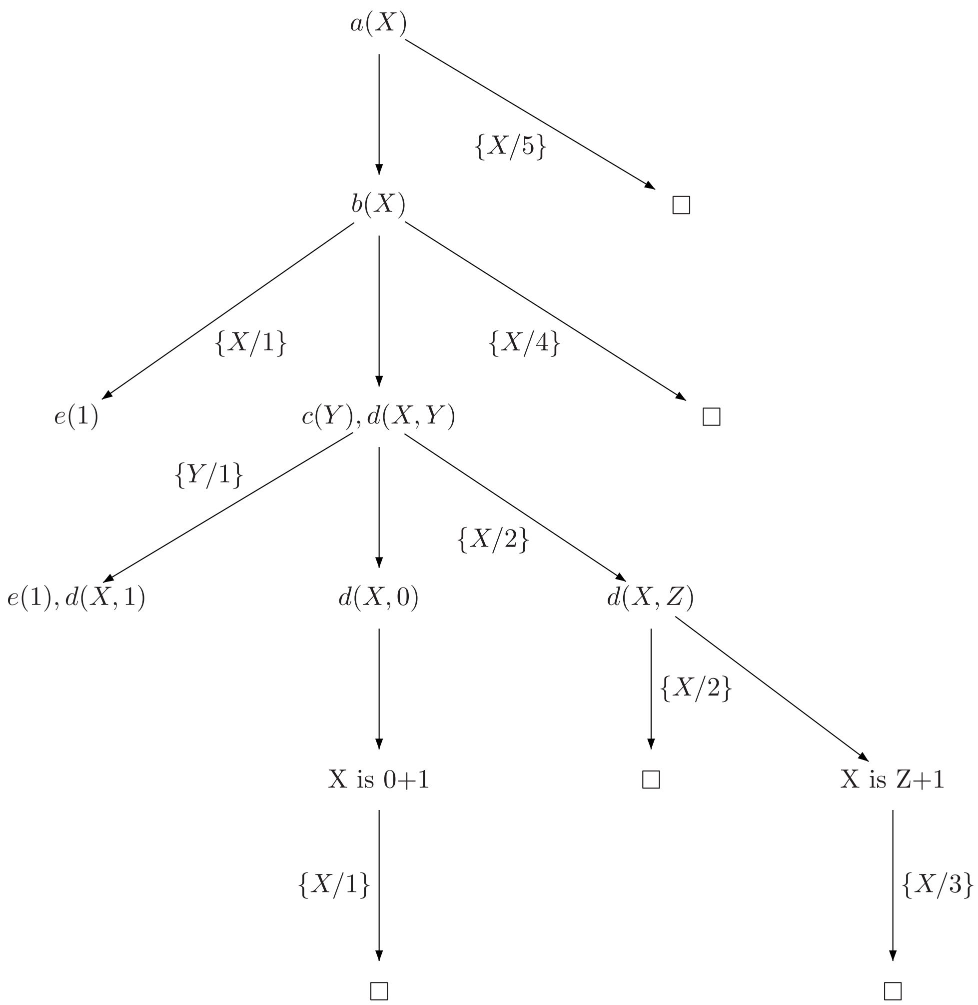
Genau Bedeutung des Cuts

Falls Anfrage ?-$A_1, \ldots, A_k$ mit Programmklausel $B : -C_1, \ldots, C_i, !, C_{i+1}, \ldots, C_n, A_2, \ldots, A_k$ resolviert wird und der Beweis von inst. Teilzielen $C_1, \ldots, C_i$ gelingt, dann entsteht folgender SLD-Baum.

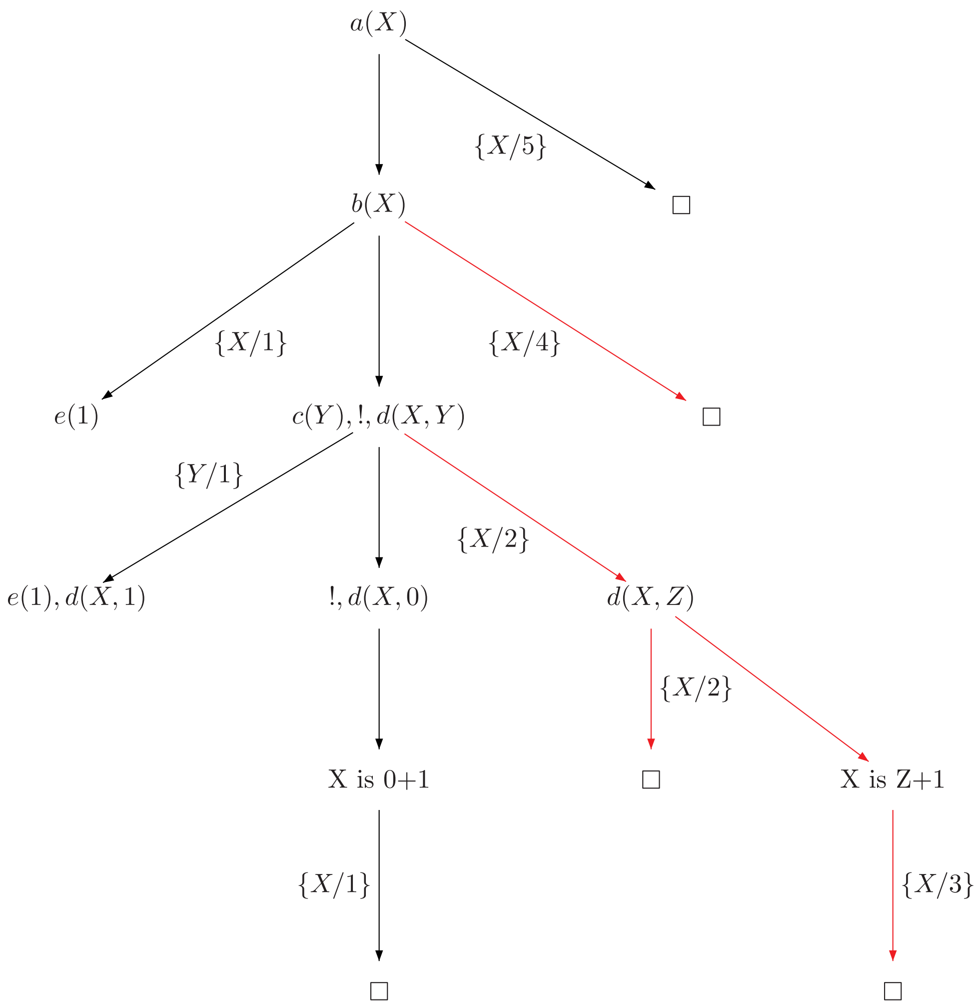


Beispiel 5.4.4

```
a(5).
b(1):-e(1).
b(X):-c(Y),d(X,Y).
b(4).
c(1):-e(1).
c(0).
c(2).
d(X,X).
d(X,Y):-X is Y+1.
e(0).
```



Antworten: X=0;X=1;X=2;X=3;X=4;X=5



Cut beeinflusst nur den Beweis der Atome, die vor dem ! in der Klausel stehen (d.h. vor b(X) und c(Y)). Die Beweise von a(X),d(X,Y) werden nicht beeinflusst.
Antworten X=0,X=1,X=5

Beispiele zur Verwendung des Cuts

```
ggT(X,0,X).
ggT(0,X,X).
ggT(X,Y,Z):- X=<Y,X>0,Y1 is Y-X,ggT(X,Y1,Z).
ggT(X,Y,Z):- Y<X,Y>0,X1 is X-Y,ggt(X1,Y,Z).

Besser:

ggT(X,0,X):-!.
ggT(0,X,X):-!.
ggT(X,Y,Z):- X=<Y,!,Y1 is Y-X,ggT(X,Y1,Z).
ggT(X,Y,Z):- X1 is X-Y,ggt(X1,Y,Z).
```

- Falls eine der ersten beiden Klauseln verwendet wird, sollte man die anderen ggT-Klauseln nicht mehr betrachten.

- Falls in der 3. Klausel `X=<Y` gelingt, dann betrachte 4. Klausel nicht mehr.
- Da wir nur $X, Y \geq 0$ betrachten, erreicht man die 3. und 4. Klausel nur bei $X > 0, Y > 0$ (wegen ! in Klauslen 1 und 2) $\Rightarrow$ lasse $X > 0, Y > 0$ weg.
- lasse $Y < X$ in Klausel 4 weg (wegen ! in Klausel 3).

`remove(X,Xs,Ys)` wahr falls die Liste Ys aus Xs entsteht, indem man alle Vorkommen von X gelöscht werden.
`?-remove(1,[0,1,2,1],Ys).` $\Rightarrow$ Ys=[0,2].

```
remove(_,[],[]).
remove(X,[X|Xs],Ys):- !,remove(X,Xs,Ys).
remove(X,[Y|Xs],Ys):- remove(X,Xs,Ys).
```

Ohne Cut: `?-remove(1,[0,1,2,1],Ys).` $\Rightarrow$ Ys=[0,2]; Ys=[0,2,1]; Ys=[0,1,2]; Ys=[0,1,2,1].

VL 27.06.2006

5.4.1 Meta-Variablen und Negation

Bislang: Terme: $f(t_1, \ldots, t_n), X$ (f Funktionssymbol, t_i Term, X Variable die mit Term instantiiert werden kann). Atomare Formel: $p(t_1, \ldots, t_n)$, (p Prädikatssymbol, t_i Term).
Jetzt: Keine Trennung mehr zwischen Funktions-/Prädikatssymbolen, d.h. zwischen Termen/atomaren Formeln.

Meta-Variablen Variablen, die für <u>Formeln</u> statt für <u>Terme</u> stehen.

Meta-Prädikate Prädikatssymbole, die <u>Formeln</u> (statt <u>Terme</u>) als Argument haben.

Beispiel 5.4.2.1

```
p(a). <- 1-stelliges Meta-Prädikatssymbol
a.    <- ist ein 0-stelliges Prädikatssymbol

?-p(X),X.
```

$$\begin{tikzcd} \text{p(X),X} \arrow[d, "\{X/a\}"] \\ \text{a} \arrow[d] \\ \square \end{tikzcd}$$

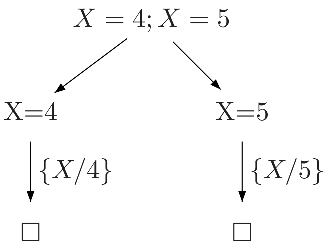
Antwort: X=a
Resolution + Unifikation wie bisher.
Aber: Meta-Variablen müssen instantiiert sein, bevor sie zur Resolution verwendet werden.
`?-p(X),X,Y.`
$\Rightarrow$ Programmfehler, denn irgendwann müsste man `?-Y.` beweisen (mit uninstantiierter Meta-Variable Y).

Meta-Prädikate sind z.B. nützlich, um logische Junktoren zu programmieren.

```
or(X,Y):-X.
or(X,Y):-Y.
```

Vordefiniert in Prolog:

```
;(X,Y):-X.
;(X,Y):-Y.
:-op(1100,xfy,;) Disjunktion
:-op(1000,xfy,,) Konjunktion

?-X=4;X=5.
```



Antwort: X=4;X=5

```
p(X,Y):-X=1,Y=1;X=2,Y=2.
```

Antwort: (X=1,Y=1);(X=2,Y=2)

Beispiel 5.4.2.2

```
if(A,B,C) entspricht if A then B else C

if(A,B,C):-A,!,B.
if(A,B,C):-C.

?-if(A,B,C).
```

Beweist zunächst A. Wenn Beweis von A gelingt, dann muss B bewiesen werden. Wenn B scheitert, dann scheitert auch `if(A,B,C)` denn wegen des `!` kann die zweite Klausel nicht verwendet werden.
Ohne Cut: `if(A,B,C)` wahr, falls (A und B wahr) oder C wahr. Wenn Beweis von A nicht gelingt, dann muss C bewiesen werden.
Vordefiniert in Prolog: "`if(A,B,C)`"="A→B;C"

Beispiel 5.4.2.3

Aus Programm jetzt nicht nur existenzquantifizierte Konjunktionen $A_1 \wedge \ldots \wedge A_k$ (A_i atomare Teilformel) herleiten, sondern auch Konjunkionen, die negierte atomare Formeln $\neg A$ enthalten. $\mathcal{P} \models \neg A$ gilt **nie**! Struktur die alle atomaren Formeln wahr macht, ist Modell von $\mathcal{P}$, aber nicht von $\neg A$. $\Rightarrow$ Prädikat "not" kann nicht die Semantik der normalen Negation haben. Bei der Realisation der Negation werden zwei Annahmen getroffen:

(a) Aus dem Programm sind alle wahren Aussagen über die Welt herleitbar (Closed-World Assumption). Falls A nicht herleitbar ist, dann ist A auch nicht wahr. $\Rightarrow \neg A$ wahr.

(b) Falls eine Aussage aus dem Programm nicht herleitbar ist, dann wir das in endliche Zeit festgestellt.

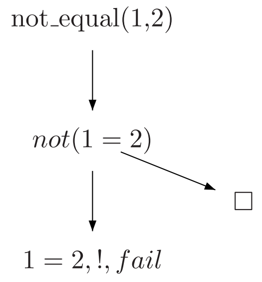
⇒ Interpretiere die Negation “endlichen Fehlschlag“. Um $\neg A$ zu beweisen, versuche A zu beweisen, falls dies in endlicher Zeit fehlschlägt, dann ist der Beweis von $\neg A$ erfolgreich.

```
not(A):-A,!,fail. (fail ist 0-stellig vordef. und schlägt immer fehl)
not(A).
```

not/1 ist vordefiniert, auch als Präfix-Schreibweise “`\+`“

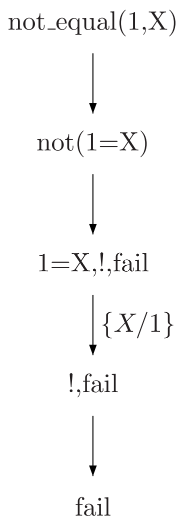
Beispiel 5.4.2.4

```
not_equal(X,Y):-not(X=Y).

?-not_equal(1,2).
```



Antwort: Yes.

```
?-not_equal(1,1).
No.

?-X=2,not_equal(1,X).
Yes.

?-not_equal(1,X).
```



⇒ Variablen in negierten Anfragen sind **allquantifiziert** (“Gilt $1 \neq X$“ für alle X).

Problem wenn Annahme (b) nicht zutrifft.

```
even(0).
even(X):-X1 is X-2,even(X1).
```

“`even(1)`“ folgt **nicht** aus Programm. Aber Fehlschlag wird nicht in endlicher Zeit festgestellt (Nichtterminierung).
Problem wenn (a) nicht zutrifft.

```
even(0).
even(X):-X>=2,X1 is X-2,even(X1).

?-not(even(1)).
Yes.

?-not(even(-2)). <- Programm enthält nicht alles Wissen. CWA trifft nicht zu
Yes.
```

Korrekte Version

```
even(0):-!.
even(X):-X>0,!,X1 is X-1,not(even(X1)).
even(0):-X1 is X+1,not(even(X1)).
```

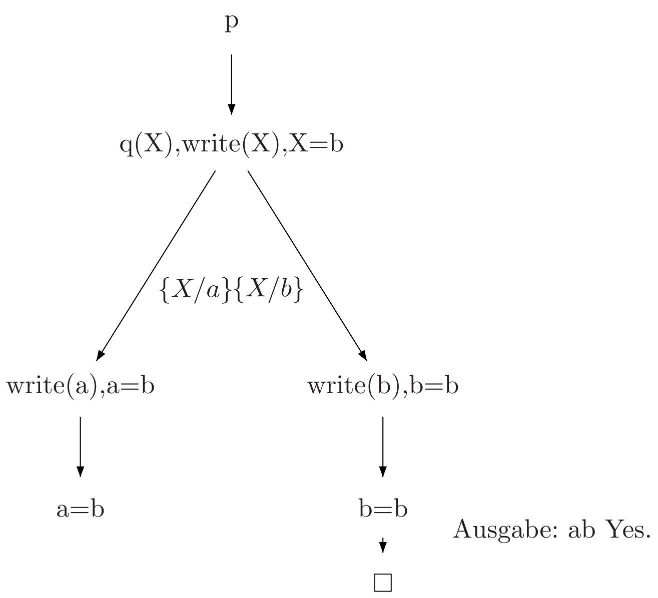
5.5 Ein- und Ausgabe

Bisher:
Eingabe: Anfragen an das Programm
Ausgabe: Antwortsubstitution, Yes/No.
Jetzt: Vordefinierte Prädikate die Ein-/Ausgabe mit Seiteneffekten druchführen.
write/1: Schreibt Argumentterm in den aktuellen Ausgabestream (standartmäßig Bildschirm des Benutzers)
`?-write(t).`gelingt immer, aber als Seiteneffekt wird t ausgegeben.
`?-X is 2+3,write(X).`
Antwort: X=5.
Seiteneffekt: 5 auf Bildschirm.

```
?-write('Dies ist eine Konstante').
Dies ist eine Konstante
Yes.

mult(X,Y):- Ergebnis is X*Y,write(X*Y),write(' = '),write(Ergebnis).

?-mult(3,4).
3*4 = 12.
```

Achtung: Beim Rücksetzen werden Seiteneffekte nicht rückgängig gemacht.

```
q(a).
q(b).
p:-q(X),write(X),X=b.

?-p.
```


p
q(X),write(X),X=b
{X/a}{X/b}
write(a),a=b
write(b),b=b
a=b
b=b
Ausgabe: ab Yes.
□