Logische und funktionale Programmierung

Vorlesung 11: Logikprogramme

Babeş-Bolyai Universität, Department für Informatik, Cluj-Napoca
csacarea@cs.ubbcluj.ro

19. Dezember 2016

Wiederholung: Horn-Klauseln

Definition

Eine Klausel K ist eine Hornklausel gdw. sie höchstens ein positives Literal enthält (d.h., höchstens eines ihrer Literale ist eine atomare Formel und die anderen Literale sind negierte atomare Formeln). Eine Hornklausel heißt negativ, falls sie nur negative Literale enthält (d.h., falls sie die Gestalt $\{\neg A_1, \ldots, \neg A_k\}$ für atomare Formeln $A_1, \ldots, A_k$ hat). Eine Hornklausel heißt definit, falls sie ein positives Literal enthält (d.h., falls sie die Gestalt $\{B, \neg C_1, \ldots, \neg C_n\}$ für atomare Formeln $B, C_1, \ldots, C_n$ hat).

- Eine Menge definiter Hornklauseln entspricht einer Konjunktion von Implikationen
- $\{\{p, \neg q\}, \{\neg r, \neg p, s\}, \{s\}\}$ ist äquivalent zur Formel

$$((p \vee \neg q) \wedge (\neg r \vee \neg p \vee s) \wedge s)$$

und damit zur folgenden Formel

$$(q \rightarrow p) \wedge (r \wedge p \rightarrow s) \wedge s).$$

Zusammenhang zur Logikprogrammierung

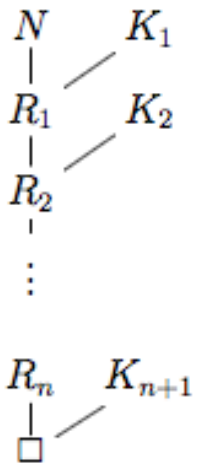
- Fakten sind definite Hornklauseln ohne negative Literale (d.h., sie enthalten genau ein positives Literal). Ein Beispiel ist die Klausel $\{s\}$. In der Logikprogrammierung schreibt man: `s.`
- Regeln sind definite Hornklauseln mit negativen Literalen. Ein Beispiel ist die Klausel $\{\neg r, \neg p, s\}$. In der Logikprogrammierung schreibt man: `s :- r, p.`
- Anfragen sind negative Hornklauseln. Ein Beispiel ist die Klausel $\{\neg p, \neg q\}$. In der Logikprogrammierung schreibt man: ?- p,q.

SLD-Resolution

Definition

Sei $\mathcal{K}$ eine Hornklauselmenge 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 $\square$ ist aus der Klausel K in $\mathcal{K}^n$ durch SLD-Resolution herleitbar gdw. es eine Folge von Klauseln $K_1, \ldots, K_m$ gibt, so dass $K_1 = K \in \mathcal{K}^n$ und $K_m = \square$ ist und so dass für alle $2 \leq i \leq m$ gilt: K_i ist ein Resolvent von K_{i-1} und einer Klausel aus K_d.

Man erkennt sofort, dass in einer SLD–Resolution alle Klauseln $K_1, \ldots, K_m$ negativ sind. SLD–Resolutionen haben also die folgende Gestalt:



Hierbei sind $K_1, \ldots, K_{n+1} \in \mathcal{K}^d$ definite Hornklauseln aus der Eingabemenge, $N \in \mathcal{K}^n$ ist eine negative Hornklausel aus der Eingabemenge und die Resolventen $R_1, \ldots, R_n$ sind ebenfalls negative Hornklauseln.

Korrektheit und Vollständigkeit der SLD-Resolution

Sei $\mathcal{K}$ eine Menge von Hornklauseln. Dann ist $\mathcal{K}$ unerfüllbar gdw. $\square$ aus einer negativen Klausel N in $\mathcal{K}$ durch SLD - Resolution herleitbar ist.

Syntax und Semantik von Logikprogrammen

- Die Reihenfolge der Literale in einer Klausel und die Reihenfolge der Klauseln in einer Klauselmenge spielt jetzt eine Rolle.
- Wir betrachten Folgen statt Mengen.
- Klausel = Folgen von Literalen.
- Klauselmengen = Folgen von Klauseln.
- Eine Klausel kann ein Literal auch mehrfach enthalten und eine Klauselmenge kann eine Klausel mehrfach enthalten.

Syntax von Logikprogrammen

Definition

Eine nichtleere endliche Menge $\mathcal{P}$ von definiten Hornklauseln über einer Signatur (Σ, Δ) heißt Logikprogramm über (Σ, Δ). Die Klauseln aus $\mathcal{P}$ werden auch Programmklauseln genannt und man unterscheidet die folgenden Arten von Programmklauseln:

- *Fakten (oder* Tatsachenklauseln*) sind Klauseln der Form $\{B\}$, wobei B eine atomare Formel ist*
- *Regeln (oder* Prozedurklauseln*) sind Klauseln der Form $\{B, \neg C_1, \ldots, \neg C_n\}$ mit $n \geq 1$. Hierbei sind B und $C_1, \ldots, C_n$ atomare Formeln. Der Aufruf eines Logikprogramms geschieht durch eine*
- *Anfrage (oder* Zielklausel*) G der Form $\{\neg A_1, \ldots, \neg A_k\}$ mit $k \geq 1$.*

- Eine Klausel ist die allquantifizierte Disjunktion ihrer Literale und eine Klauselmenge (wie z.B. ein Logikprogramm) repräsentiert die Konjunktion ihrer Klauseln.
- Beim Aufruf eines Logikprogramms $\mathcal{P}$ mit der Anfrage $G = \{\neg A_1, \ldots, \neg A_k\}$ soll folgendes bewiesen werden:

$$\mathcal{P} \models \exists X_1, \ldots, X_p A_1 \wedge \cdots \wedge A_k$$

  Hierbei sind $X_1, \ldots, X_p$ die Variablen in G.
- Die Variablen in den Programmklauseln sind implizit allquantifiziert und die Variablen in den Anfragen sind implizit existenzquantifiziert!

- Die Folgerbarkeitsbeziehung

$$\mathcal{P} \models \exists X_1, \ldots, X_p A_1 \wedge \cdots \wedge A_k$$

ist äquivalent zur Unerfüllbarkeit der Klauselmenge $\mathcal{P} \cup \{G\}$, d.h., zur Unerfüllbarkeit von $\mathcal{P} \cup \{\forall X_1, \ldots, X_p \neg A_1 \vee \cdots \vee \neg A_k\}$.

- Die Unerfüllbarkeit von $\mathcal{P} \cup \{G\}$ ist äquivalent dazu, dass es eine endliche Teilmenge von Grundinstanzen der Klauseln aus $\mathcal{P} \cup \{G\}$ gibt, die ebenfalls unerfüllbar ist. Diese Menge kann nicht nur aus definiten Klauseln bestehen, d.h., sie enthält also auch mindestens eine Grundinstanz von G.

- $\mathcal{P} \models \exists X_1, \ldots, X_p \; A_1 \wedge \cdots \wedge A_k$ ist äquivalent dazu, dass es eine Menge von Grundtermen $t_1, \ldots, t_p$ existiert, so dass

$$\mathcal{P} \cup \{(\neg A \vee \cdots \vee \neg A_k)[X_1/t_1, \ldots, X_p/t_p]\}$$

unerfüllbar ist bzw, dass

$$\mathcal{P} \models A_1 \wedge \cdots \wedge A_k[X_1/t_1, \ldots, X_p/t_p].$$

- Bei der Auswertung von Logikprogrammen ist das Ziel, nicht nur die Folgerbarkeitsbeziehung

$$\mathcal{P} \models \exists X_1, \ldots, X_p \, A_1 \wedge \cdots \wedge A_k$$

zu untersuchen, sondern man möchte auch die Grundterme $t_1, \ldots, t_p$ berechnen, die die gültigen Lösungen für die Anfrage darstellen.
- Substitutionen, bei denen die gewünschte Anfrage aus den Programmklauseln folgt, bezeichnet man als Antwortsubstitution. Hierbei schränkt man die Substitution auf die Variablen der Anfrage ein.
- Falls die Antwortsubstitution die Variablen der Anfrage durch Terme mit Variablen ersetzt, so können die verbleibenden Variablen durch beliebige Terme ersetzt werden.
- Die Antwortsubstitutionen werden im Lauf des SLD-Resolutionsbeweises erzeugt, wenn die leere Klausel $\square$ hergeleitet wird.

Beispiel

```
mutterVon(renate,susanne).
verheiratet(gerd,renate).
vaterVon(V,K) :- verheiratet(V,F), mutterVon(F,K).
```

Wenn wir dieses Logikprogramm $\mathcal{P}$ als Klauselmenge schreiben, ergibt sich

$$\{ \quad \{\mathsf{mutterVon}(\mathsf{renate}, \mathsf{susanne})\}, \\ \{\mathsf{verheiratet}(\mathsf{gerd}, \mathsf{renate})\}, \\ \{\mathsf{vaterVon}(V, F), \neg\mathsf{verheiratet}(V, F), \neg\mathsf{mutterVon}(F, K)\} \quad \}.$$

Beispiel

Wir untersuchen die Anfrage

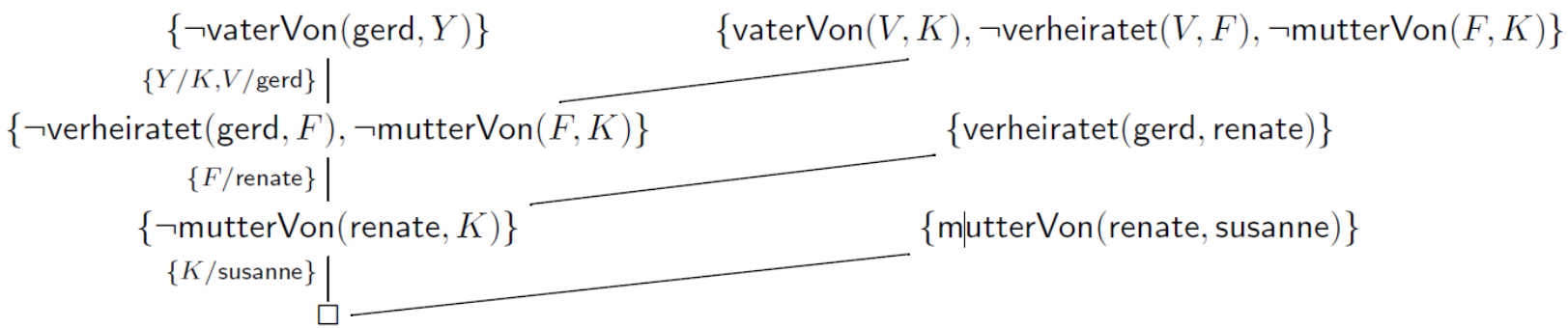
```
?- vaterVon(gerd,Y).
```

Dies bedeutet, dass wir zu der obigen Menge von definiten Hornklauseln noch die negative Hornklausel G

$$\{\neg \mathsf{vaterVon}(\mathsf{gerd}, Y)\}$$

hinzufügen. Wir erhalten nun den folgenden SLD–Resolutionsbeweis, um die leere Klausel herzuleiten. Hierbei wurde in jedem Resolutionsschritt der mgu angegeben, der auf die negative Elternklausel und die (ggf. variablenumbenannte) Programmklausel angewandt wurde.

Beispiel


{¬vaterVon(gerd, Y)}
{vaterVon(V, K), ¬verheiratet(V, F), ¬mutterVon(F, K)}
{Y/K, V/gerd}
{¬verheiratet(gerd, F), ¬mutterVon(F, K)}
{verheiratet(gerd, renate)}
{F/renate}
{¬mutterVon(renate, K)}
{mutterVon(renate, susanne)}
{K/susanne}
□

Beispiel

Die Antwortsubstitution ergibt sich nun, indem man die einzelnen Substitutionen komponiert

$$\{K/\mathsf{susanne}\} \circ \{F/\mathsf{renate}\} \circ \{Y/K, V/\mathsf{gerd}\} = \{K/\mathsf{susanne}, F/\mathsf{renate}, Y/\mathsf{susanne}, V/\mathsf{gerd}\}$$

und dann auf die Variablen der Anfrage einschränkt. Da in unserem Beispiel die Anfrage nur die Variable Y enthält, ergibt sich also die Antwortsubstitution $\{Y/\mathsf{susanne}\}$.

Semantik der Logikprogrammierung

- Wir definieren nun die Semantik von Logikprogrammen.
- Hierbei werden wir drei verschiedene Möglichkeiten vorstellen, die Semantik festzulegen und wir werden beweisen, dass alle drei Möglichkeiten äquivalent sind.
- Diese Arten werden als deklarative, prozedurale und Fixpunkt-Semantik bezeichnet.

Deklarative Semantik der Logikprogrammierung

- Die Idee der deklarativen (oder modelltheoretischen) Semantik ist, das Logikprogramm als statische Datenbank aufzufassen und die wahren Aussagen über das Programm mit Hilfe der Folgerbarkeit der Prädikatenlogik zu definieren.
- Genauer definieren wir jeweils die Semantik eines Programms $\mathcal{P}$ bezüglich einer Anfrage G.
- Die deklarative Semantik besteht aus allen Grundinstanzen von G, die *wahre Aussagen* über das Programm $\mathcal{P}$ sind.

Deklarative Semantik eines Logikprogramms

Definition

Sei $\mathcal{P}$ ein Logikprogramm und $G = \{\neg A_1, \ldots, \neg A_k\}$ eine Anfrage. Hierbei sind $A_1, \ldots, A_k$ also atomare Formeln. Dann ist die deklarative Semantik von $\mathcal{P}$ bezüglich G definiert als

$$D[[\mathcal{P}, G]] = \{\sigma(A_1 \wedge \cdots \wedge A_k) \mid \mathcal{P} \models \sigma(A_1 \wedge \cdots \wedge A_k), \sigma \text{ ist Grundsubstitutio}$$

Jede Grundinstanz $\sigma(A_1 \wedge \cdots \wedge A_k)$ in $D[[\mathcal{P}, G]]$ enthält als Lösung die entsprechende Grundsubstitution der Variablen aus $A_1, \ldots, A_k$.

Beispiel

- Wir betrachten wieder das Logikprogramm $\mathcal{P}$ und die Anfrage $G = \{\neg \text{vaterVon}(\text{gerd}, Y)\}$.
- Die einzige Grundinstanz von vaterVon(gerd, Y), die aus $\mathcal{P}$ folgt, ist `vaterVon(gerd, susanne)` (d.h. $\mathcal{P} \models \mathit{vaterVon}(\mathit{gerd}, \mathit{susanne})$). Somit gilt

$$D[[\mathcal{P}, G]] = \{\mathit{vaterVon}(\mathit{gerd}, \mathit{susanne})\}.$$

- Würde $\mathcal{P}$ noch das zusätzliche Faktum `mutterVon(renate, peter)` enthalten, so ergäbe sich

$$D[[\mathcal{P}, G] = \{\mathit{vaterVon}(\mathit{gerd}, \mathit{susanne}), \mathit{vaterVon}(\mathit{gerd}, \mathit{peter})\}.$$

Prozedurale Semantik der Logikprogrammierung

- Die prozedurale (oder operationelle) Semantik *operationalisiert* die deklarative Semantik, indem explizit angegeben wird, wie die entsprechenden Folgerungen aus $\mathcal{P}$ berechnet werden.
- Hierzu wird SLD-Resolution verwendet, wobei jeweils mitprotokolliert wird, welche Substitutionen auf die Variablen angewandt werden.
- Genauer geben wir einen abstrakten Interpreter für Logikprogramme an, der auf sogenannten Konfigurationen operiert.

Prozedurale Semantik der Logikprogrammierung

- Eine Konfiguration ist ein Paar aus einer Anfrage (d.h., einer negativen Klausel) und einer Substitution.
- Hierbei starten wir mit der Konfiguration $(G, \emptyset)$ aus der ursprünglichen Anfrage und der identischen Substitution $\emptyset$.
- Das Ziel ist, schließlich eine Endkonfiguration der Gestalt $(\square, \sigma)$ zu erreichen.
- In diesem Fall ist σ (bzw. die Einschränkung von σ auf die Variablen der ursprünglichen Anfrage G) die gefundene *Antwortsubstitution*.
- Eine *Berechnung* ist eine Folge von Konfigurationen, wobei bei der Definition des Interpreters festgelegt wird, welche Konfigurationsübergänge erlaubt sind.

Bemerkung

- Die bei Logikprogrammen verwendete Form der SLD-Resolution unterscheidet sich in drei Punkten von der allgemeinen SLD-Resolution, die wir besprochen haben.
- Anstatt Variablenumbenennungen jeweils auf beide Elternklauseln anzuwenden, schränken wir uns auf die sogenannte standardisierte SLD-Resolution ein.
- Hier dürfen Variablenumbenennungen nur auf die (definiten) Programmklauseln angewendet werden und nicht auf die jeweils andere (negative) Elternklausel.
- Eine solche Einschränkung ist natürlich generell ohne Beschränkung der Allgemeinheit möglich.

Bemerkung

- Im Resolutionsschritt wird jeweils nur zwischen zwei Literalen resolviert, nicht zwischen beliebig vielen.
- Wir verwenden also nur binäre SLD-Resolution.
- Wie schon gezeigt wurde, erhält diese Einschränkung aber auf Hornklauselmengen die Vollständigkeit.

Bemerkung

- Schließlich betrachten wir Klauseln nicht mehr als Mengen, sondern als Folgen von Literalen. Dies bedeutet also, dass in einer solchen Folge ein Literal also auch mehrfach auftreten kann.
- Beispielsweise ergibt dann die Resolution der Klauseln $\{\neg p, \neg p\}$ und $\{p\}$ den Resolventen$\{\neg p\}$.
- Hierbei gilt $p \in \Delta_0$.

Prozedurale Semantik eines Logikprogramms $\mathcal{P}$

- *Eine* Konfiguration *ist ein Paar* (G, σ)*, wobei* G *eine Anfrage oder die leere Klausel* $\square$ *ist und wobei* σ *eine Substitution ist.*
- *Es gibt einen* Rechenschritt $(G_1, \sigma_1) \vdash_{\mathcal{P}} (G_2, \sigma_2)$ *gdw.*
  - $G_1 = \{\neg A_1, \ldots, \neg A_k\}$ *mit* $k \geq 1$
  - *es existiert eine Programmklausel* $K \in \mathcal{P}$ *und eine Variablenumbenennung* ν *mit* $\nu(K) = \{B, \neg C_1, \ldots, \neg C_n\}$ *und* $n \geq 0$*, so dass*
    - G_1 *und* $\nu(K)$ *keine gemeinsamen Variablen haben*
    - *es ein* $1 \leq i \leq k$ *gibt, so dass* A_i *und* B *mit einem mgu* σ *unifizierbar sind*
  - $G_2 = \sigma(\{\neg A_1, \ldots, \neg A_{i-1}, \neg C_1, \ldots, \neg C_n, \neg A_{i+1}, \ldots, \neg A_k\})$
  - $\sigma_2 = \sigma \circ \sigma_1$

Prozedurale Semantik eines Logikprogramms $\mathcal{P}$

- *Eine* Berechnung *von $\mathcal{P}$ bei Eingabe von $G = \{\neg A_1, \ldots, \neg A_k\}$ ist eine (endliche oder unendliche) Folge von Konfigurationen der Form*

$$(G, \varnothing) \vdash_{\mathcal{P}} (G_1, \sigma_1) \vdash_{\mathcal{P}} (G_2, \sigma_2) \vdash_{\mathcal{P}} \ldots$$

- *Eine mit $(\square, \sigma)$ terminierende Berechnung, die mit $(G, \varnothing)$ startet (wobei $G = \{\neg A_1, \ldots, \neg A_k\}$), heißt* erfolgreich *mit dem Rechenergebnis $\sigma(A_1 \wedge \ldots \wedge A_k)$. Die berechnete* Antwortsubstitution *ist σ eingeschränkt auf die Variablen aus G.*

Damit ist die prozedurale Semantik *von $\mathcal{P}$ bezüglich G definiert als*

$$P[\![\mathcal{P}, G]\!] = \{\sigma'(A_1 \wedge \ldots \wedge A_k) \mid (G, \varnothing) \vdash_{\mathcal{P}}^{+} (\square, \sigma), \\ \sigma'(A_1 \wedge \ldots \wedge A_k) \text{ ist Grundinstanz von } \sigma(A_1 \wedge \ldots \wedge A_k)\}.$$

Hierbei steht "$\vdash_{\mathcal{P}}^{+}$" für die transitive Hülle von "$\vdash_{\mathcal{P}}$" (d.h., es gilt $(G, \varnothing) \vdash_{\mathcal{P}}^{+} (\square, \sigma)$ gdw. $(G, \varnothing) \vdash_{\mathcal{P}} \ldots \vdash_{\mathcal{P}} (\square, \sigma)$). Analog dazu definieren wir auch für alle $l \in \mathbb{N}$ die Relation $\vdash_{\mathcal{P}}^{l}$ als $(G, \sigma) \vdash_{\mathcal{P}}^{l} (G_l, \sigma_l)$ gdw. es G_i und σ_i gibt mit $(G, \sigma) \vdash_{\mathcal{P}} (G_1, \sigma_1) \vdash_{\mathcal{P}} \ldots \vdash_{\mathcal{P}} (G_l, \sigma_l)$.

Beispiel (s. vorheriges Bsp.)

$$
\begin{aligned}
& (\{\neg \mathsf{vaterVon}(\mathsf{gerd}, Y)\}, \varnothing) \\
\vdash_{\mathcal{P}} \quad & (\{\neg \mathsf{verheiratet}(\mathsf{gerd}, F), \neg \mathsf{mutterVon}(F, K)\}, \{Y/K, V/\mathsf{gerd}\}) \\
\vdash_{\mathcal{P}} \quad & (\{\neg \mathsf{mutterVon}(\mathsf{renate}, K)\}, \{F/\mathsf{renate}, Y/K, V/\mathsf{gerd}\}) \\
\vdash_{\mathcal{P}} \quad & (\square, \{K/\mathsf{susanne}, F/\mathsf{renate}, Y/\mathsf{susanne}, V/\mathsf{gerd}\})
\end{aligned}
$$

und die Antwortsubstitution ist $\{Y/\mathsf{susanne}\}$. *Wir erhalten*

$$P[\![\mathcal{P}, G]\!] = \{\mathsf{vaterVon}(\mathsf{gerd}, \mathsf{susanne})\}.$$

Das nächste Beispiel zeigt, dass es bei den Rechenschritten der prozeduralen Semantik noch zwei Indeterminismen gibt:

- Zum einen muss man die Programmklausel K auswählen, mit der resolviert werden soll.
- Zum anderen muss man jeweils das nächste Literal A_i aus der momentanen Anfrage auswählen, das zur Resolution verwendet werden soll.

Beispiel

$$\{ \quad \{\mathsf{p}(X,Z), \neg\mathsf{q}(X,Y), \neg\mathsf{p}(Y,Z)\}, \\ \{\mathsf{p}(U,U)\}, \\ \{\mathsf{q}(\mathsf{a},\mathsf{b})\} \ \}$$

mit $\mathsf{p},\mathsf{q} \in \Delta_2$ *und* $\mathsf{a},\mathsf{b} \in \Sigma_0$. *Die zu untersuchende Anfrage ist*

$$G = \{\neg\mathsf{p}(V,\mathsf{b})\}.$$

Beispiel

Die folgende Berechnung ist nicht erfolgreich (aber endlich und nicht weiter fortsetzbar). Hierbei ist jeweils das Literal A_i unterstrichen, das zur Resolution verwendet wurde:

$$
\begin{aligned}
& (\{\underline{\neg \mathsf{p}(V, \mathsf{b})}\}, \varnothing) \\
\vdash_{\mathcal{P}} \quad & (\{\underline{\neg \mathsf{q}(V, Y)}, \neg \mathsf{p}(Y, \mathsf{b})\}, \{X/V, Z/\mathsf{b}\}) \\
\vdash_{\mathcal{P}} \quad & (\{\underline{\neg \mathsf{p}(\mathsf{b}, \mathsf{b})}\}, \{V/\mathsf{a}, Y/\mathsf{b}\} \circ \{X/V, Z/\mathsf{b}\}) \\
\vdash_{\mathcal{P}} \quad & (\{\neg \mathsf{q}(\mathsf{b}, Y'), \underline{\neg \mathsf{p}(Y', \mathsf{b})}\}, \{X'/\mathsf{b}, Z'/\mathsf{b}\} \circ \{V/\mathsf{a}, Y/\mathsf{b}\} \circ \{X/V, Z/\mathsf{b}\}) \\
\vdash_{\mathcal{P}} \quad & (\{\neg \mathsf{q}(\mathsf{b}, \mathsf{b})\}, \{U/\mathsf{b}, Y'/\mathsf{b}\} \circ \{X'/\mathsf{b}, Z'/\mathsf{b}\} \circ \{V/\mathsf{a}, Y/\mathsf{b}\} \circ \{X/V, Z/\mathsf{b}\})
\end{aligned}
$$

Beispiel

Eine erfolgreiche Berechnung wäre hingegen die folgende, die die gleichen drei ersten Schritte hat, aber dann mit der zweiten anstelle der ersten Programmklausel resolviert:

$$
\begin{aligned}
& (\{\overline{\neg \mathsf{p}(V, \mathsf{b})}\}, \varnothing) \\
\vdash_{\mathcal{P}} \quad & (\{\overline{\neg \mathsf{q}(V, Y)}, \neg \mathsf{p}(Y, \mathsf{b})\}, \{X/V, Z/\mathsf{b}\}) \\
\vdash_{\mathcal{P}} \quad & (\{\overline{\neg \mathsf{p}(\mathsf{b}, \mathsf{b})}\}, \{V/\mathsf{a}, Y/\mathsf{b}\} \circ \{X/V, Z/\mathsf{b}\}) \\
\vdash_{\mathcal{P}} \quad & (\square, \underbrace{\{U/\mathsf{b}\} \circ \{V/\mathsf{a}, Y/\mathsf{b}\} \circ \{X/V, Z/\mathsf{b}\}}_{\{U/\mathsf{b},\, V/\mathsf{a},\, Y/\mathsf{b},\, X/\mathsf{a},\, Z/\mathsf{b}\}})
\end{aligned}
$$

Die Antwortsubstitution ist also $\{V/\mathsf{a}\}$ *und somit gilt* $\mathsf{p}(\mathsf{a}, \mathsf{b}) \in P[\![\mathcal{P}, G]\!]$.

Beispiel

In diesem Beispiel gibt es aber noch eine weitere erfolgreiche Berechnung, indem man bereits im ersten Schritt gleich mit der zweiten Programmklausel resolviert:

$$\begin{array}{rl} & (\{\underline{\neg \mathsf{p}(V, \mathsf{b})}\}, \varnothing) \\ \vdash_{\mathcal{P}} & (\square, \{U/\mathsf{b}, V/\mathsf{b}\}) \end{array}$$

Die Antwortsubstitution ist nun $\{V/\mathsf{b}\}$ *und es gilt daher auch* $\mathsf{p}(\mathsf{b}, \mathsf{b}) \in P[\![\mathcal{P}, G]\!]$.

- Die deklarative und die prozedurale Semantik sind äquivalent!

Fixpunkt-Semantik der Logikprogrammierung

Bei der folgenden dritten Art der Semantik–Definition wird die Semantik eines Logikprogramms als Fixpunkt einer Transformation $\underline{\text{trans}}_{\mathcal{P}}$ definiert. Diese Transformation leitet nach und nach alle wahren Aussagen über ein Programm her. Im Unterschied zur deklarativen Semantik ist die Semantik hier nicht modelltheoretisch definiert, sondern es wird (ähnlich wie bei der prozeduralen Semantik) die Einschränkung auf Hornklauseln ausgenutzt, um durch resolutionsähnliche Schritte wahre Aussagen herzuleiten. Im Unterschied zur prozeduralen Semantik wird bei der Herleitung aber nicht von der Anfrage ausgegangen, sondern man geht nur vom Programm aus und leitet alle wahren Aussagen (ohne Variablen) her.

Fixpunkt-Semantik der Logikprogrammierung

Zu einem Logikprogramm $\mathcal{P}$ ist $\underline{\text{trans}}_{\mathcal{P}}$ eine Funktion, die Mengen von Aussagen in Mengen von Aussagen überführt. Hierbei enthält $\underline{\text{trans}}_{\mathcal{P}}(M)$ zusätzlich zu M alle weiteren Aussagen, die aus M in einem Resolutionsschritt mit Hilfe des Programms $\mathcal{P}$ herleitbar sind. Im Folgenden bezeichnet $Pot(\mathcal{A}t(\Sigma, \Delta, \varnothing))$ die Menge aller Mengen von atomaren Formeln ohne Variablen.

Die Transformation $\underline{\text{trans}}_{\mathcal{P}}$

Sei $\mathcal{P}$ ein Logikprogramm über einer Signatur (Σ, Δ). Die dazugehörige Transformation $\underline{\text{trans}}_{\mathcal{P}}$ ist eine Funktion

$$\underline{\text{trans}}_{\mathcal{P}} : Pot(\mathcal{A}t(\Sigma, \Delta, \varnothing)) \to Pot(\mathcal{A}t(\Sigma, \Delta, \varnothing))$$

mit

$$\underline{\text{trans}}_{\mathcal{P}}(M) = M \cup \{A' \mid \{A', \neg B'_1, \ldots, \neg B'_n\} \textit{ ist Grundinstanz einer Klausel } \{A, \neg B_1, \ldots, \neg B_n\} \in \mathcal{P} \textit{ und } B'_1, \ldots, B'_n \in M\}$$

Die Idee zur Definition der Semantik besteht nun darin, zuerst von der leeren Menge von Aussagen $\varnothing$ auszugehen. Dann beschreibt $\varnothing$ alle Aussagen, die in null Schritten aus $\mathcal{P}$ herleitbar sind und $\underline{\text{trans}}_{\mathcal{P}}(\varnothing)$ sind alle Aussagen (ohne Variablen), die in höchstens einem Resolutionsschritt mit $\mathcal{P}$ herleitbar sind. Es handelt sich also um alle Grundinstanzen der Fakten von $\mathcal{P}$. Analog dazu sind $\underline{\text{trans}}_{\mathcal{P}}(\underline{\text{trans}}_{\mathcal{P}}(\varnothing)) = \underline{\text{trans}}^2_{\mathcal{P}}(\varnothing)$ alle Aussagen, die in höchstens zwei Resolutionsschritten aus $\mathcal{P}$ herleitbar sind, etc. Die Menge

$$M_{\mathcal{P}} = \varnothing \cup \underline{\text{trans}}_{\mathcal{P}}(\varnothing) \cup \underline{\text{trans}}^2_{\mathcal{P}}(\varnothing) \cup \underline{\text{trans}}^3_{\mathcal{P}}(\varnothing) \cup \ldots = \bigcup_{i \in \mathbb{N}} \underline{\text{trans}}^i_{\mathcal{P}}(\varnothing)$$

enthält also alle Aussagen ohne Variablen, die aus $\mathcal{P}$ herleitbar sind. Man kann die Semantik von $\mathcal{P}$ bezüglich einer Anfrage G dadurch definieren, dass man alle Grundinstanzen von G betrachtet, die in $\bigcup_{i \in \mathbb{N}} \underline{\text{trans}}^i_{\mathcal{P}}(\varnothing)$ enthalten sind.

Beispiel

$$\{ \quad \{\mathsf{mutterVon}(\mathsf{renate}, \mathsf{susanne})\},$$
$$\{\mathsf{verheiratet}(\mathsf{gerd}, \mathsf{renate})\},$$
$$\{\mathsf{vaterVon}(V, K), \neg\mathsf{verheiratet}(V, F), \neg\mathsf{mutterVon}(F, K)\} \quad \}.$$

Es ergibt sich

$$\begin{aligned} \underline{\mathrm{trans}}_{\mathcal{P}}(\varnothing) &= \{\mathsf{mutterVon}(\mathsf{renate}, \mathsf{susanne}), \mathsf{verheiratet}(\mathsf{gerd}, \mathsf{renate})\} \\ \underline{\mathrm{trans}}^2_{\mathcal{P}}(\varnothing) &= \underline{\mathrm{trans}}_{\mathcal{P}}(\varnothing) \cup \{\mathsf{vaterVon}(\mathsf{gerd}, \mathsf{susanne})\} \end{aligned}$$

und $\underline{\mathrm{trans}}^i_{\mathcal{P}}(\varnothing) = \underline{\mathrm{trans}}^2_{\mathcal{P}}(\varnothing)$ *für alle* $i \geq 2$. *In diesem Beispiel ist* $M_{\mathcal{P}}$ *also nach nur 2 Iterationsschritten erreicht.*

Beispiel

Im allgemeinen kann die Iteration zur Berechnung von $M_{\mathcal{P}}$ aber unendlich sein.

```
p(a).
p(f(X)) :- p(X).
```

Es entspricht der folgenden Klauselmenge:

$$\{ \quad \{\mathsf{p}(\mathsf{a})\}, \\ \quad \{\mathsf{p}(\mathsf{f}(X)), \neg\mathsf{p}(X)\} \ \}$$

Nun gilt

$$\begin{aligned} \underline{\mathrm{trans}}_{\mathcal{P}}(\varnothing) &= \{\mathsf{p}(\mathsf{a})\}, \\ \underline{\mathrm{trans}}^2_{\mathcal{P}}(\varnothing) &= \{\mathsf{p}(\mathsf{a}), \mathsf{p}(\mathsf{f}(\mathsf{a}))\}, \\ \underline{\mathrm{trans}}^3_{\mathcal{P}}(\varnothing) &= \{\mathsf{p}(\mathsf{a}), \mathsf{p}(\mathsf{f}(\mathsf{a})), \mathsf{p}(\mathsf{f}^2(\mathsf{a}))\}, \textit{etc.} \end{aligned}$$

Somit ergibt sich $\underline{\mathrm{trans}}^{i+1}_{\mathcal{P}}(\varnothing) = \{\mathsf{p}(\mathsf{a}), \mathsf{p}(\mathsf{f}(\mathsf{a})), \ldots, \mathsf{p}(\mathsf{f}^i(\mathsf{a}))\}$ *und* $M_{\mathcal{P}} = \bigcup_{i\in\mathbb{N}} \underline{\mathrm{trans}}^i_{\mathcal{P}}(\varnothing) = \{\mathsf{p}(\mathsf{f}^i(\mathsf{a})) \mid i \in \mathbb{N}\}$.

Die obige Konstruktion berechnet einen *Fixpunkt* der Transformation $\underline{\text{trans}}_{\mathcal{P}}$, denn es gilt $\underline{\text{trans}}_{\mathcal{P}}(M_{\mathcal{P}}) = M_{\mathcal{P}}$. In der Tat handelt es sich hierbei sogar um einen speziellen Fixpunkt von $\underline{\text{trans}}_{\mathcal{P}}$. $M_{\mathcal{P}}$ ist nämlich der *kleinste* Fixpunkt von $\underline{\text{trans}}_{\mathcal{P}}$, wenn man Mengen bezüglich der Teilmengenrelation $\subseteq$ vergleicht. Dies bedeutet, dass $M_{\mathcal{P}}$ nur die Aussagen enthält, die wirklich unbedingt aufgenommen werden müssen, wenn man alle aus $\mathcal{P}$ herleitbaren Aussagen bekommen will.

Wir zeigen nun, dass die oben berechnete Menge $M_{\mathcal{P}}$ wirklich stets der kleinste Fixpunkt von $\underline{\text{trans}}_{\mathcal{P}}$ ist. Dies beweist auch, dass $\underline{\text{trans}}_{\mathcal{P}}$ stets einen (eindeutigen) kleinsten Fixpunkt hat.

Wie üblich bezeichnet man jede transitive und antisymmetrische Relation als *Ordnung*. Die Relation $\subseteq$ auf $Pot(\mathcal{A}t(\Sigma, \Delta, \varnothing))$ ist offensichtlich eine reflexive Ordnung, d.h., es gilt für alle $M_1, M_2, M_3 \subseteq \mathcal{A}t(\Sigma, \Delta, \varnothing)$:

- $M_1 \subseteq M_1$ (Reflexivität)
- Aus $M_1 \subseteq M_2$ und $M_2 \subseteq M_3$ folgt $M_1 \subseteq M_3$ (Transitivität)
- Aus $M_1 \subseteq M_2$ und $M_2 \subseteq M_1$ folgt $M_1 = M_2$. (Antisymmetrie)

- Wenn man die Folge von Mengen

$$\emptyset, \underline{transp}_{\mathcal{P}}(\emptyset), \underline{transp}^2_{\mathcal{P}}(\emptyset), \underline{transp}^3_{\mathcal{P}}(\emptyset), \ldots$$

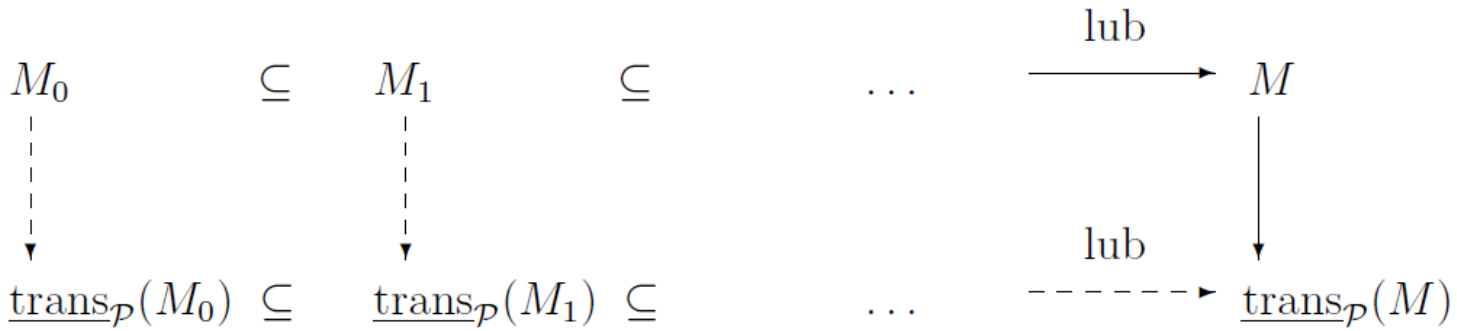
  betrachtet, sieht man, dass diese Mengen offensichtlich immer größer bezüglich der Ordnung $\subseteq$ werden.
- Solche Folgen bezeichnet man als Ketten.
- Die gesuchte Menge $M_{\mathcal{P}}$ ist dann gerade der *Grenzwert* dieser Kette, d.h., es ist ihre kleinste obere Schranke.

- Eine reflexive Ordnung ist vollständig, falls sie ein kleinstes Element hat und falls jede Kette eine kleinste obere Schranke besitzt.
- Solch eine Ordnung wird auch als *complete partial order* oder *cpo* bezeichnet.
- $Pot(\mathcal{A}t(\Sigma, \Delta, \emptyset))$ ist vollständig.
- Zu jedem Programm $\mathcal{P}$ existiert also die kleinste obere Schranke $M_{\mathcal{P}}$ der obigen Kette

Die Funktion $\underline{\text{trans}}_{\mathcal{P}}$ besitzt zwei wichtige Eigenschaften, die entscheidend dafür sind, dass sie tatsächlich immer einen kleinsten Fixpunkt besitzt. Sie ist *monoton* und *stetig*.

- Monotonie bedeutet in diesem Zusammenhang, dass sich aus größren Mengen von Aussagen auch mehr Aussagen herleiten lassen.
- Stetigkeit bedeutet, dass $\underline{trans}_{\mathcal{P}}$ den Grenzwert jeder Kette auf das gleiche Ergebnis abbildet, das man erhalten würde, wenn man das Ergebnis für jedes individuelle Element der Kette berechnen würde und dann den Grenzwert dieser Ergebnisse bildet.

Stetigkeit von $\underline{trans_{\mathcal{P}}}$



Stetigkeit von $\underline{trans}_{\mathcal{P}}$

- Wenn man zunächst die Kette $M_0 \subseteq M_1 \subseteq \ldots$ betrachtet, ihren Grenzwert $M = \bigcup_{i \in \mathbb{N}} M_i$ bildet und dann $\underline{trans}_{\mathcal{P}}$ darauf anwendet (durchgezogene Pfeile), so muss sich dasselbe ergeben, wie wenn man erst $\underline{trans}_{\mathcal{P}}$ auf die einzelnen Elemente der Kette anwendet und dann den Grenzwert bildet (gestrichelte Pfeile).

Lemma: Monotonie und Stetigkeit

(a) *Die Funktion* $\underline{\text{trans}}_{\mathcal{P}}$ *ist* monoton, *d.h., falls* $M_1 \subseteq M_2$ *ist, so gilt auch* $\underline{\text{trans}}_{\mathcal{P}}(M_1) \subseteq \underline{\text{trans}}_{\mathcal{P}}(M_2)$.

(b) *Die Funktion* $\underline{\text{trans}}_{\mathcal{P}}$ *ist* stetig, *d.h., für jede Kette*

$$M_0 \subseteq M_1 \subseteq M_2 \subseteq \ldots$$

gilt $\underline{\text{trans}}_{\mathcal{P}}(\bigcup_{i \in \mathbb{N}} M_i) = \bigcup_{i \in \mathbb{N}} \underline{\text{trans}}_{\mathcal{P}}(M_i)$.

Beweis

Beweis. Teil (a) ist offensichtlich. Wir zeigen nun Teil (b). Hierbei folgt $\underline{\text{trans}}_{\mathcal{P}}(\bigcup_{i\in\mathbb{N}} M_i) \supseteq \bigcup_{i\in\mathbb{N}} \underline{\text{trans}}_{\mathcal{P}}(M_i)$ aus der Monotonie. Der Grund ist, dass wegen der Monotonie $\underline{\text{trans}}_{\mathcal{P}}(M_i) \subseteq \underline{\text{trans}}_{\mathcal{P}}(\bigcup_{i\in\mathbb{N}} M_i)$ für alle i gilt.

Um auch $\underline{\text{trans}}_{\mathcal{P}}(\bigcup_{i\in\mathbb{N}} M_i) \subseteq \bigcup_{i\in\mathbb{N}} \underline{\text{trans}}_{\mathcal{P}}(M_i)$ zu zeigen, sei $A' \in \underline{\text{trans}}_{\mathcal{P}}(\bigcup_{i\in\mathbb{N}} M_i)$. Dann ist $\{A', \neg B'_1, \ldots, \neg B'_n\}$ Grundinstanz einer Klausel $\{A, \neg B_1, \ldots, \neg B_n\} \in \mathcal{P}$ und $B'_1, \ldots, B'_n \in \bigcup_{i\in\mathbb{N}} M_i$. Da $M_0 \subseteq M_1 \subseteq M_2 \subseteq \ldots$ eine Kette ist, gibt es daher ein $j \in \mathbb{N}$ mit $B'_1, \ldots, B'_n \in M_j$. Damit folgt auch $A' \in \underline{\text{trans}}_{\mathcal{P}}(M_j) \subseteq \bigcup_{i\in\mathbb{N}} \underline{\text{trans}}_{\mathcal{P}}(M_i)$. □

- Nun können wir zeigen, dass $\underline{trans}_{\mathcal{P}}$ in der Tat immer einen kleinsten Fixpunkt hat und dass dieser gerade der kleinsten oberen Schranke der Kette

$$\emptyset, \underline{transp}_{\mathcal{P}}(\emptyset), \underline{transp}^2_{\mathcal{P}}(\emptyset), \underline{transp}^3_{\mathcal{P}}(\emptyset), \ldots$$

entspricht.

Fixpunktsatz

Für jedes Logikprogramm $\mathcal{P}$ besitzt $\underline{trans}_{\mathcal{P}}$ einen kleinsten Fixpunkt $lfp(\underline{trans}_{\mathcal{P}})$. Hierbei gilt $lfp(\underline{trans}_{\mathcal{P}}) = \bigcup_{i \in \mathbb{N}} \underline{trans}^i_{\mathcal{P}}$.

Fixpunkt-Semantik eines Logikprogramms

Definition

Sei $\mathcal{P}$ ein Logikprogramm und $G = \{\neg A_1, \ldots, \neg A_k\}$ eine Anfrage. Hierbei sind $A_1, \ldots, A_k$ also atomare Formeln. Dann ist die Fixpunkt-Semantik von $\mathcal{P}$ bezüglich G definiert als

$$\mathcal{F}[[\mathcal{P}, G]] = \{\sigma(A_1 \wedge \cdots \wedge A_k) \mid \sigma(A_i) \in \mathit{lfp}(\underline{\mathit{trans}}_{\mathcal{P}}) \text{ für alle } i\}.$$

Satz: Äquivalenz der Fixpunkt-Semantik zu den anderen Semantiken

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