Prof. Dr. Christoph Beierle, Dr. Harald Ganzinger, Prof. Dr. Michael Hanus

Kurs 01816

Logisches und funktionales Programmieren

LESEPROBE

Fakultät für
Mathematik und Informatik

FernUniversität in Hagen

1.3 Beweisstrategie

Die prinzipielle Aufgabe eines Prolog-Systems ist es, Literale in Anfragen aufgrund vorgegebener Fakten und Regeln zu beweisen. Zur Konstruktion solcher Beweise wird eine bestimmte Strategie verwendet, die wir in diesem Abschnitt kennenlernen werden. Zunächst wird das allgemeine Resolutionsprinzip vorgestellt, mit dem man immer einen Beweis finden kann, falls überhaupt einer existiert (die theoretischen Hintergründe hierfür werden im Rahmen dieses Kurses nicht behandelt; genauere Informationen kann man im anfangs angegebenen Buch von Lloyd oder im Kurs 1695 *Deduktions- und Inferenzsysteme* finden). Aus Effizienzgründen wird in Prolog-Systemen jedoch eine eingeschränkte Strategie verwendet, die wir im Anschluß erläutern.

1.3.1 Das Prinzip der Resolution

Die Semantik von Regeln haben wir oben wie folgt definiert:

> Wenn jedes der Literale $\texttt{L}_1$, ..., $\texttt{L}_n$ beweisbar ist und
>
> $\texttt{L}_0$ `:-` $\texttt{L}_1$`,...,`$\texttt{L}_n$`.`
>
> eine Regel ist, dann ist auch das Literal $\texttt{L}_0$ beweisbar.

Diese wichtige Schlußregel ist in der mathematischen Logik schon lange unter der Bezeichnung *modus ponens* oder *Abtrennungsregel* bekannt. Wenn wir Fakten als Regeln mit leerer rechter Seite (n = 0) auffassen, dann erhält durch diese Schlußregel ein Prolog-Programm (Menge von Fakten und Regeln) eine Bedeutung. Eine Umformulierung dieser Schlußregel ergibt eine Methode zum Beweisen von Literalen. Diese Methode heißt *Resolutionsprinzip* :

> Ist
>
> $\texttt{L}_0$ `:-` $\texttt{L}_1$`,...,`$\texttt{L}_n$`.`
>
> eine Regel, dann ist das Literal $\texttt{L}_0$ beweisbar, wenn jedes der Literale $\texttt{L}_1$, ..., $\texttt{L}_n$ beweisbar ist.

Obwohl das Resolutionsprinzip und die Abtrennungsregel in ihren Aussagen äquivalent sind, haben sie eine unterschiedliche Sichtweise. Das Resolutionsprinzip besagt, wie man vorgegebene Literale beweisen kann: Indem eine passende Regel gesucht wird, deren linke Seite mit dem vorgegebenen Literal übereinstimmt, und dann versucht wird, die Literale auf der rechten Seite der Regel zu beweisen. Allerdings gibt es im allgemeinen keine Regel, die direkt zu dem vorgegebenen Literal paßt. Wenn wir das Literal `member(X, [a,b])` bzgl. der o.a. Klauseln für `member` beweisen wollen, dann gibt es keine Klausel, deren linke Seite mit dem Literal übereinstimmt. Es gibt aber die Klausel

```
member(E,[E|_]).
```

die „passen könnte“: Über die Bedeutung von Variablen hatten wir gesagt, daß an ihrer Stelle beliebige andere Objekte eingesetzt werden dürfen. Wenn wir also die Variablen `X` und `E` durch das Atom `a` und die anonyme Variable durch die Liste `[b]` ersetzen, dann paßt das Faktum zu dem vorgegebenen Literal, und nach Anwendung des Resolutionsprinzips ist das Literal bewiesen (die rechte Regelseite ist ja bei Fakten leer). Zu beachten ist jedoch, daß das vorgegebene Literal nicht in der vollen Allgemeinheit beweisbar ist, sondern nur, wenn die Variable `X` durch das Atom `a` ersetzt wird. Genau dies wird auch vom Prolog-System ausgegeben.

Um also das Resolutionsprinzip auf Prolog-Programme anzuwenden, muß der Prozeß des Ersetzens von Variablen berücksichtigt werden. Wenn wir zusätzlich noch berücksichtigen, daß wir im allgemeinen nicht ein einzelnes Literal, sondern eine Menge von Literalen zu beweisen haben, erhalten wir das *allgemeine Resolutionsprinzip*:

Der Beweis der Anfrage

$$\textit{?- } \boldsymbol{A_1,\ldots,A_{i-1},A_i,A_{i+1},\ldots,A_m}.$$

läßt sich zurückführen auf den Beweis der Anfrage

$$\text{?- } \sigma(A_1,\ldots,A_{i-1},L_1,\ldots,L_n,A_{i+1},\ldots,A_m)\ .$$

wenn

$$\mathtt{L_0\ :-\ L_1,\ldots,L_n}.$$

eine Regel ist, deren Variablen verschieden sind von denen in $A_1,\ldots,A_m$ (dies kann man durch Umbenennung der in dieser Regel verwendeten Variablen immer erreichen), und wenn gilt:

1. Die Funktion σ ersetzt nur Variablen durch neue Terme, wobei gleiche Variablen auch durch gleiche Terme ersetzt werden. Alles andere (Konstanten, Funktoren) wird durch σ nicht verändert.
2. Die Literale $\sigma(A_i)$ und $\sigma(\mathtt{L_0})$ sind identisch.

Mit diesem allgemeinen Resolutionsprinzip können alle Anfragen bewiesen werden, die bzgl. der eingegebenen Fakten und Regeln beweisbar sind. Ein *Resolutionsbeweis* ist dabei eine Folge von Anfragen, wobei in jedem Schritt das allgemeine Resolutionsprinzip angewendet wird und die letzte Anfrage keine Literale mehr enthält.

Beispiel: Gegeben seien die Klauseln für das Prädikat `member` :

```
member(E,[E|_]).
member(E,[_|R]) :- member(E,R).
```

Die folgenden Anfragen bilden einen Resolutionsbeweis:

```
?- member(norbert,[heinz,fritz,herbert,hubert,norbert,andreas]).
?- member(norbert,[fritz,herbert,hubert,norbert,andreas]).
?- member(norbert,[herbert,hubert,norbert,andreas]).
?- member(norbert,[hubert,norbert,andreas]).
?- member(norbert,[norbert,andreas]).
?- .
```

In den ersten vier Schritten wurde die zweite Klausel angewendet, während im letzten Schritt die erste Klausel (Faktum) angewendet worden ist. Jedes Prolog-System arbeitet nach diesem Resolutionsprinzip, wobei am Ende eines Beweises *yes* oder *no* (Beweis gefunden bzw. nicht gefunden) oder die Terme, die anstelle der Variablen in der Anfrage im Verlauf des Beweises eingesetzt wurden, ausgegeben werden. Die beiden Klauseln

```
last([E],E).
last([_|R],E) :- last(R,E).
```

definieren ein Prädikat `last`, das die Relation zwischen einer Liste und dem letzten Element dieser Liste beschreibt. Am Ende des folgenden Resolutionsbeweises wird vom Prolog-System $X{=}c$ ausgegeben:

```
?- last([a,b,c],X).
?- last([b,c],X).
?- last([c],X).
?- .
```

Bevor wir uns genauer mit der Frage beschäftigen, nach welcher Strategie ein Prolog-System die einzelnen Klauseln ausprobiert, gehen wir auf das Problem ein, wie man in einem Resolutionsschritt die Funktion σ findet, die zwei Literale durch Ersetzung von Variablen in Übereinstimmung bringt.

1.3.2 Unifikation

In einem Resolutionsschritt müssen die Variablen in einem Literal und einer Klausel derart durch neue Terme ersetzt werden, daß das Literal und die linke Seite der Klausel übereinstimmen. Dieser Prozeß der Ersetzung von Variablen heißt *Unifikation*. Die Unifikation berechnet Funktionen, die Variablen durch neue Terme ersetzen. Solche Funktionen heißen Substitutionen: Eine *Substitution* kann auf beliebige Terme, Literale, Klauseln und Anfragen angewendet werden und ersetzt dabei gewisse Variablen durch neue Terme, wobei gleiche Variablen auch durch gleiche Terme ersetzt werden und alles andere gleich bleibt. Insbesondere verändert eine Substitution nicht die Namen von Prädikaten oder Funktoren und löscht auch keine Atome oder Strukturen. Da jede Substitution nur bestimmte Variablen verändert und alles andere gleich läßt, werden Substitutionen durch die Veränderung dieser Variablen charakterisiert. So bezeichnet

σ = {N/mueller, J/23}

die Substitution, die jedes Vorkommen der Variablen N durch das Atom mueller und jedes Vorkommen der Variablen J durch die Zahl 23 ersetzt. Also gilt

σ(person(christine,N,datum(12,7,J)))
= person(christine,mueller,datum(12,7,23))

Weiterhin definieren wir:

> Sind T_1 und T_2 zwei Terme oder Literale, dann heißt eine Substitution σ *Unifikator* für T_1 und T_2, wenn gilt: $\sigma(T_1) = \sigma(T_2)$. Zwei Terme oder Literale heißen *unifizierbar*, falls ein Unifikator für sie existiert. σ heißt *allgemeinster Unifikator* für T_1 und T_2 , wenn für jeden anderen Unifikator σ' für T_1 und T_2 gilt: Es existiert eine Substitution ϕ, so daß $\sigma' = \phi \circ \sigma$ gilt, d.h. σ' ist die Komposition von σ und ϕ (zuerst σ, dann ϕ).

Aus allgemeinsten Unifikatoren kann man also alle anderen Unifikatoren als Spezialfälle ableiten. Allgemeinste Unifikatoren haben die Eigenschaft, nur die Variablen durch andere Terme zu ersetzen, für die das unbedingt notwendig ist, um die beiden Terme in Übereinstimmung zu bringen. Alle anderen Variablen bleiben unverändert. So ist die Substitution

σ_1 = {Tag/3, Monat/4, M/4, J/83}

zwar ein Unifikator für die Terme datum(Tag,Monat,83) und datum(3,M,J), aber kein allgemeinster, weil er für den Monat einen konkreten Wert (4) einsetzt, was aber nicht notwendig ist. Dagegen ist

σ_2 = {Tag/3, Monat/M, J/83}

ein allgemeinster Unifikator für diese Terme.

Bei der Anwendung des allgemeinen Resolutionsprinzips ist es wichtig, die Unifikatoren nicht zu speziell zu wählen, da man sonst beim Beweisen zu leicht in Sackgassen laufen kann. Glücklicherweise existieren für Prolog-Terme und Literale immer allgemeinste Unifikatoren, falls sie unifizierbar sind. Es gibt sogar eine effektive Methode, um sie zu berechnen: