6 Logische Programmiersprachen

Der Begriff von „Rechnen“ in verschiedenen Programmiersprachen basiert auf unterschiedlichen Vorgehensweisen:

- Imperative Programmiersprachen: Folge von Variablenmanipulationen (Zuweisungen)
- Funktionale Programmiersprachen: Folge von Ersetzungen (Reduktion zur Normalform)
- Logische Programmiersprachen: Folge von Beweisschritten (in einem logischen Kalkül), d.h. es wird versucht zu beweisen, ob eine Aussage wahr bzgl. der eingegebenen Regeln ist.

Im Folgenden werden wir alle Beispiele in der Programmiersprache **Prolog** angeben. Prolog ist zwar schon eine recht alte Sprache (die Ursprünge gehen zurück auf das Jahr 1972), aber es ist ein Standard in der logischen Programmierung und als ISO-Standard auch industriell relevant. Die Notation ist im Gegensatz zu Haskell mehr logikorientiert, d.h. statt `(`$f\ \ e_1 \dots e_n$`)` schreibt man in Prolog f`(`$e_1, \dots, e_n$`)`.

6.1 Einführung

Zunächst geben wir eine kurze Einführung in die Struktur logischer Programme.

Ein **Programm** ist eine Menge von Prädikaten (Relationen).

Ein **Prädikat** ist definiert durch Fakten und Implikationen/Regeln. Daher nennt man dies manchmal auch **regelorientierte Programmierung**, die für Anwendungen in der KI (Künstliche Intelligenz) wichtig ist.

Ein **Literal**[1] ist die Anwendung eines Prädikats auf Argumente (Terme). Intuitiv entspricht dies einer Aussage.

Beispiel 6.1 (Familienbeziehungen)**.** Wir wollen Familienbeziehungen als logisches Programm modellieren. Die betrachteten Prädikate (Aussagen) sind dabei:

- Vater: `vater(v,k)` :⇔ `v` ist Vater von `k`
- Großvater: `grossvater(g,e)` :⇔ `g` ist Großvater von `e`

[1] Dieser Begriff aus der Logikprogrammierung sollte nicht mit Literalen, d.h. festen Werten, aus der imperativen Programmierung verwechselt werden.

Diese Definitionen bilden das folgende Prolog-Programm:

```
vater(fritz,thomas).  % Fritz ist Vater von Thomas.
vater(thomas,maria).
vater(thomas,anna).

grossvater(G,E) :- vater(G,V), vater(V,E).
% G Großvater von E, falls G Vater von V und V Vater von E ist.
```

Es folgen einige kurze Erläuterungen zur Syntax von Prolog:

Kommentare beginnen mit `%` und erstrecken sich bis zum Zeilenende.

Variablennamen beginnen mit einem Großbuchstaben (`G,V,E`).

Literale haben die Form „`<Prädikatname>(<arg1>,...,<argn>)`“. Beachte dabei, dass zwischen dem Prädikatnamen und der danach öffnenden Klammer *kein* Leerzeichen stehen darf.

Faktum Ein Faktum hat die Form „`<Literal>.`“ und muss mit einem Punkt am Ende abgeschlossen werden. Ein Faktum repräsentiert eine Aussage, die immer wahr ist.

Implikation Eine Implikation hat die Form „`<Lit> :- <Lit`$_1$`>,...,<Lit`$_n$`>.`“. Das Symbol „`:-`“ kann man als Implikation “$\Leftarrow$” lesen, und jedes Komma entspricht einer Konjunktion “$\wedge$”. Wichtig ist auch hier, dass die Regel mit einem Punkt am Ende abgeschlossen wird. Intuitiv bedeutet diese Regel: „Falls `<Lit`$_1$`>` und ... und `<Lit`$_n$`>` wahr sind, dann ist auch `<Lit>` wahr.“

Terme Die Argumente von Literalen sind **Terme** (dies entspricht Ausdrücken oder Datentermen in funktionalen Sprachen ohne definierte Funktionen). Diese sind zusammengesetzt aus Variablen, Konstanten, Zahlen und Funktoren.

Funktor Die **Funktoren** entsprechen Datenkonstruktoren in funktionalen Sprachen, wie z. B. `datum(1,januar,J)`. Hier ist „`datum`“ ein Funktor.

Listen sind wie in Haskell definiert, allerdings ist die Notation etwas anders: In Prolog schreibt man `[H|T]` statt `(h:t)` bzw. `[E1,E2,E3|T]` statt `(e1:e2:e3:t)`, aber auch `[]` oder `[1,2,3]` für Listen fester Länge.

Klausel Eine **Klausel** ist ein Faktum oder eine Regel/Implikation.

Logikprogramme entsprechen logischen Formeln, wobei die Variablen in Klauseln universell quantifiziert sind. Zum Beispiel entspricht die Regel

```
grossvater(G,E) :- vater(G,V), vater(V,E)
```

der logischen Formel

$$\forall G.\forall E.\forall V.\ (grossvater(G,E) \leftarrow vater(G,V) \wedge vater(V,E))$$

Eine **Anfrage** ist eine Konjunktion von Literalen. Intuitiv bedeutet eine Anfrage, dass wir daran interessiert sind, ob diese Literale logisch aus dem Programm folgen. In Anfragen sind auch Variablen erlaubt, in diesem Fall sollen dann Werte berechnet/erraten werden, sodass für diese Werte die Anfrage logisch aus dem Programm folgt.

Beispiel 6.2 (Anfragen)**.** Gegeben sei unser obiges Prolog-Programm. Wenn wir dies in das Prolog-System geladen haben, können wir folgende Anfragen stellen:

```
?- vater(fritz,thomas).
yes
?- vater(fritz,anna).
no
?- grossvater(fritz,anna).
yes
?- grossvater(fritz,E). % Welche Enkel E hat Fritz?
E=maria  ;              % Die Eingabe von ";" entspricht
E=anna                  % der Suche nach weiteren Lösungen.
```

Wie wir sehen, sucht das Prolog-System nach passenden Lösungen und es kann durchaus auch mehrere Lösungen geben.

Wichtig ist festzuhalten, dass es in Prolog keine festen Ein-/Ausgabeargumente gibt, d. h. Prolog kann bidirektional rechnen, weil man bei jedem Argument eine Variable in der Anfrage einsetzen kann:

```
?- grossvater (G,anna). % Welche Großväter hat Anna?
G=fritz
```

Logische Programmiersprachen haben somit folgende Unterschiede zu funktionalen Programmiersprachen:

- Es erfolgt ein Raten von passenden Werten.
- Es existiert eine (nichtdeterministische) Suche nach Lösungen.
- Es gibt keine festen Ein-/Ausgabeargumente: Mit der Definition einer Funktion hat man automatisch auch immer die Umkehrfunktion bzw. -relation zur Verfügung.

Analog zur funktionalen Programmierung können wir in Prolog auch musterorieniert programmieren. Als Beispiel betrachten wir ein Prädikat zur Listenkonkatenation:

`append(L1,L2,L3)` :⇔ `L3` ist Konkatenation von `L1` und `L2`

Mit ähnlichen Überlegungen wie bei dem entsprechenden funktionalen Programm können wir zu folgender Definition kommen:

```
append([]    , L , L     ).
append([X|Xs], Ys, [X|Zs]) :- append(Xs, Ys, Zs).
```

An diesem Beispiel können wir schon ein Schema zur Übersetzung funktionaler Programme in relationale Programme erkennen:

- Füge ein zusätzliches Ergebnisargument hinzu: Eine n-stellige Funktion wird übersetzt in ein $(n+1)$-stelliges Prädikat.
- Herausziehen geschachtelter Funktionsaufrufe: Aus `X=f(`$\underbrace{\texttt{f(Y)}}_{\texttt{Z}}$`)` wird das `Z` herausgezogen, wir erhalten `f(Y,Z), f(Z,X)`. Dabei ist `Z` eine neu eingeführte Variable.

Die relationale Version ist allerdings flexibler einsetzbar als die funktionale, wie man an folgenden Beispielen sehen kann.

Beispiel 6.3 (Listenoperationen)**.**

- Präfix einer Liste abspalten:

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

- Letztes Element einer Liste berechnen:

```
?- append(_,[X],[1,2,3,4]).
X=4
```

  Hier bezeichnet „_“ eine anonyme Variable, an deren Wert man nicht interessiert ist.