6.2 Programmierung mit Logik

Bei Programmieren mit Sprachen der Logik wird das Programm als ein System von **Tatsachen** und **Schlussfolgerungen** aufgefasst.

Eingabe:

- eine Menge von **Fakten** (= wahre Aussagen, = gültige Prädikate)
- eine Menge von **Regeln** (zur Ableitung neuer Fakten aus den gegebenen)

Verarbeitung:

- Feststellung, ob gestellte **Fragen** als wahr oder falsch zu beantworten sind.

Unterschied zwischen prozedural und prädikativ

- In **prozeduralen** Programmiersprachen wird ein gegebenes Problem dadurch gelöst, dass der Programmierer eine Folge von Anweisungen formuliert, deren Ausführung die Lösung des Problems bringt.
- In **prädikativen** Programmiersprachen (wie zu Beispiel den Logik-basierten Sprachen) formuliert der Anwender sein eigenes Wissen über ein gegebenes Problem, und der Computer versucht, mit Hilfe dieses Wis-sens selbstständig eine Lösung des Problems herzuleiten.

Die Sprache Prolog

Die Sprache **Prolog** („programming with logic“) wurde in den achtziger Jahren zur Programmierung mit Logik entwickelt und hat sich damals schnell verbreitet. Es gab Pläne, sehr große Systeme in Prolog zu implementieren, insbesondere umfassende Expertensysteme.

Literatur: W.F. Clocksin/C.S. Mellish: “Programming in Prolog”. Springer-Verlag, 2003.

Beispiel in Prolog (Fakultätsberechnung)

Regeln:

- `fac(0,1) :- !.` // Fakultät von 0 ist 1.
- `fac(N,F) :- N1 is N-1, fac(N1,F1), F is F1 * N!`

  // Fakultät von N ist Fakultät von (N-1) multipliziert mit N.

Frage:

- `? - fac(5,X).` // Was ist die Fakultät von 5? D. h. für
  // welche Zahl ist das Prädikat "**`fac(5,X)`**"
  // erfüllt?

Systemantwort:

- `X = 120.`

Fakten in Prolog

Ein **Faktum** ist als Element einer Relation aufzufassen:

$$\texttt{name}(o_1, o_2, \ldots, o_n).$$

`name` ist dabei eine Bezeichnung für eine Beziehung (**Prädikat**)

o_i sind Objekte (**konstante Argumente**), zwischen denen die Beziehung definiert ist.

Dabei gilt:

- Beziehungen und Objekte beginnen mit Kleinbuchstaben.
- Die Reihenfolge der Objekte ist relevant.
- Die Anzahl der Objekte ist beliebig groß, aber fest für ein bestimmtes Faktum.

Beispiele für Fakten

- `verheiratet(mann,frau).`
- `istKind(kind,mutter).`
- `wertvoll(gold).`
- `weiblich(maria).`
- `vater(hans,eva).`
- `schenkt(hans,maria,buch).`

Eine Menge von Fakten in Prolog heißt **Datenbank**.

Fragen in Prolog

Eine **Frage** ist als Test auf die Existenz eines bestimmten Elements einer Relation aufzufassen:

```
? - name(o1, ..., on).
```

Falls das gefragte Faktum herleitbar ist, gibt das System "**`yes`**" aus, ansonsten "**`no`**".

Beispiele für Fakten und Fragen

Fakten:

```
hatgern(josef,fisch).
hatgern(josef,maria).
hatgern(hans,buch).
hatgern(maria,buch).
```

Fragen:

```
? - hatgern(josef,maria).
"yes"
? - hatgern(josef,gold).
"no"
? - hatgern(maria,josef).
"no"
```

Variablen in Prolog (1)

Eine **Variable** beginnt mit einem Großbuchstaben oder mit "_".

```
? - hatgern(josef,X).
```

Prolog durchläuft für die Beantwortung der Frage sequentiell die Datenbank. Nach jeder Antwort wird angehalten und das gefundene Faktum mit einem **place-marker** markiert.

Nach der Eingabe von **<;> <CR>** wird die Suche fortgesetzt. Wird kein weiteres Faktum gefunden, so lautet die Antwort "**no**".

```
X = fisch     <;> <CR>

X = maria     <;> <CR>

"no"
```

Variablen in Prolog (2)

Anmerkung:

- In Prolog werden Variablen nicht deklariert.
- Die Gültigkeit von Variablen erstreckt sich auf Regeln oder Fakten.
- Jede Variable ist zunächst ungebunden, d.h. sie steht für keinen bestimmten Typ oder Wert.

Verknüpfung von Fragen in Prolog (1)

Fragen (G_i) können mit **UND** verknüpft werden (**Konjunktion**):

```
? - G1, G2, ..., Gn.
```

Das System gibt dann nur solche Werte für die Variablen aus, für die alle Fragen gleichzeitig erfüllt sind. Kommt also eine Variable in mehreren Teilfragen vor, so muss bei einer Lösung auch der Wert an allen Stellen der gleiche sein (konsistente Ersetz-ung).

Beispiel:

```
//  Mögen hans und maria einander?
? - hatgern(hans,maria), hatgern(maria,hans).
"no"
 //Gibt es etwas, das sowohl hans als auch maria mögen?
? - hatgern(hans,X), hatgern(maria,X).
X = buch        <;> <CR>
"no"
```

Verknüpfung von Fragen in Prolog (2)

Eine **ODER**-Verknüpfung (**Disjunktion**) ist in Prolog durch folgende Konstruktion möglich:

```
? - G1; G2; ...; Gn.
```

Prinzipielle Vorgehensweise des Systems

1. Abarbeitung der Teilziele einer Frage von links nach rechts. Erst wenn das Teilziel G_i mit "`yes`" beantwortet wurde, wird das Teilziel G_{i+1} in Angriff genommen. Wird bei der Beantwortung von G_i einer Variable einen Wert zugeordnet, so gilt diese Zuordnung auch für die Teilziele $G_{i+1}, \ldots, G_n$.
2. Die Lösungssuche für jedes Teilziel erfolgt in der Datenbank von oben nach unten.
3. Kann ein Teilziel G_i nicht erfolgreich beantwortet werden, so wird die Lösungssuche mit der Strategie des **"Backtracking"** fortgesetzt, d. h. das System geht zum Teilziel G_{i-1} zurück, die dort getroffenen Variablenzuordnungen werden wieder aufgehoben und nach einer neuen Lösung von G_{i-1} gesucht.
4. Ein vom Benutzer eingegebenes Semikolon entspricht einem vom Benutzer ausgelösten Backtracking.

Regeln in Prolog (1)

Eine **Regel** ist eine Aussage über eine Gesetzmäßigkeit. Sie besagt, dass bei Gültigkeit mehrerer Fakten ein weiterer Fakt gültig ist:

$$G :- G_1, G_2, \ldots, G_n.$$

G heißt **Kopf**, der rechte Teil der Regel **Rumpf**.

Regeln in Prolog (2)

Beispiel:

Fakten

```
männlich(albert).
männlich(eduard).
weiblich(alice).
weiblich(victoria).
eltern(eduard,victoria,albert).
eltern(alice,victoria,albert).
```

Regel

```
schwestervon(X,Y) :- weiblich(X), eltern(X,M,V), eltern(Y,M,V).
```

Anfrage

```
?- schwestervon(alice,eduard).
"yes"
```

Resolution in Prolog (1)

Zum Beweis einer Anfrage `? - A1, ..., Ak, ..., An` durchläuft das System folgende Schritte:

(R1) Kann das Anfrageziel A_k mit einem Faktum **unifiziert** ("identisch gemacht") werden, so ist die Anfrage beweisbar, wenn

```
? - A1, ..., Ak, ..., An
```

beweisbar ist.

(R2) Kann A_k mit dem Kopf einer Regel `G :- G1, ..., Gn` unifiziert werden, dann ist die Aussage beweisbar, wenn

```
? - A1, ..., Ak-1, G1, ..., Gn, Ak+1, ..., An
```

beweisbar ist.

(R3) Eine Aussage ist bewiesen, wenn sie durch fortlaufende Anwendungen von (R1) und (R2) in die leere Behauptung `? - .` überführt werden kann.

Resolution in Prolog (2)

Bei der **Unifikation** wird die maximal nötige Anzahl der Variablen konsistent ersetzt (“allgemeinster Unikator”).

Beispiel:

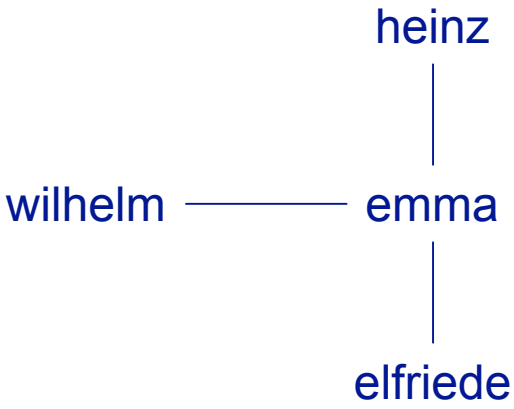
Datenbank (Wissensbasis)

```
istkind (heinz,emma).

istkind (emma,elfriede).

verheiratet(wilhelm,emma).

istvater(V,K) :- istkind(M,K), verheiratet(V,M).
```



Resolution in Prolog (3)

Anfrage

```
? - istvater(wilhelm,elfriede).
(R2) + Variablensetzung: U(V) = wilhelm, U(K) = elfriede
? - istkind(M,elfriede), verheiratet(wilhelm,M).
1. Lösungsversuch: U(M) = emma
? - istkind (emma,elfriede), verheiratet (wilhelm,emma).
(R1) ? - verheiratet (wilhelm,emma)
(R1)
? -
(R3)
"yes"
```

Weitere Konstrukte in Prolog

- Strukturen
- Rechenoperatoren
- Damit sind alle Probleme, die mit anderen Programmiersprachen lösbar sind, auch mit Prolog lösbar.

Theoretischer Hintergrund zu Prolog

- Resolution in Prolog entspricht der Resolution in der Prädikatenlogik.
- Alles, was in Prolog realisiert ist, ist in der Prädikatenlogik bewiesen worden.
- Prolog stellt letztendlich ein **Deduktionsverfahren** für Wissensbasen (Datenbanken) dar, die nur aus **Hornklauseln** bestehen. Es sind atomare Benutzerfragen möglich.