Zentralübung vom 8.1.2003

Implementierung prädikatenlogischer Formeln mit Prolog

1. Rückgriff: Prädikatenlogik

Prädikatenlogische Formeln lassen sich in Hornklauseln überführen, d.h. sie haben die Form

$$x \vee (\bar{y}_1 \vee \bar{y}_2 \vee \dots \bar{y}_n)$$

Hornklauseln können somit als Implikation geschrieben werden

$$\Rightarrow x \vee (y_1 \wedge y_2 \wedge \dots \wedge y_n)$$

und damit mit der Definition der Implikation ($a \to b \Leftrightarrow \neg a \vee b$) folgt

$$x \leftarrow (y_1 \wedge \dots \wedge y_n)$$

Diese Tatsache wird in Prolog zur Formulierung prädikatenlogischer Programme verwendet. Prolog verwendet dazu nur 2 Konstrukte:

1. Fakten: Aussagen, die wahr sind
2. Regeln: Implikation nach obigem Schema.

2. Der Prolog-Interpreter

(Literatur: U. Schöning: Logik für Informatiker)

Verwendung: gprolog.

Vorgehensweise zur Programmerstellung:

1. Editieren der Regeln und Fakten in einer Datei („Datenbank") namens <name>.pl
2. Start des Interpreters: gprolog
3. Einlesen mit: consult('<name>.pl').
4. Formulieren der Abfragen.

3. Fakten

- Einfache Fakten:

```
sunny.          /* es ist sonnig */
raining.        /* es regnet */
```

Syntaxregeln:

Fakten
- beginnen mit Kleinbuchstaben
- außer *, +, -, /, können nahezu alle Zeichen verwendet werden
- enden mit einem Punkt

Abfrage:

```
?- sunny.
yes
?- neblig.
no
```

- Fakten mit Argumenten

sind n-stellige Relationen, die mit „sprechenden" Namen versehen sind.

Beispiele:

```
age(john, 32).      /* John ist 32 Jahre alt */
age(mary, 32).      /* Mary ist 32 Jahre alt */
```

Abfragen:

```
?- age(john, 32).
yes
?- age(john, Old).
Old = 32
```

Allgemein:

- n-stellige Fakten:
  <relation>(<arg1>, <arg2>, ..., <argN>).
- Variablen: In Prolog können Variablen verwendet werden; Diese beginnen mit Großbuchstaben.

Weitere Abfragen:

- Alle Personen, deren Alter 32 ist

```
?- age(Who, 32).
Who = john ;
```

← Interpreter soll die nächste Lösung ausgeben; bei Eingabe von „a" werden alle weiteren Lösungen ausgegeben.

Zentraler Begriff in Prolog: Das Anpassen (Matchen) von Elementen der Datenbank an Variablen, nennt man Unifikation.

4. Implementierung von Regeln

Gegeben sei die Implikation

$$x \leftarrow (y_1 \wedge \dots \wedge y_n)$$

Implementierung:

Abfragen:

```
?- age(john, 32).
yes
?- age(john, Old).
Old = 32
```

Allgemein:

- n-stellige Fakten:
  <relation>(<arg1>, <arg2>, ..., <argN>).
- Variablen: In Prolog können Variablen verwendet werden; Diese beginnen mit Großbuchstaben.

Weitere Abfragen:

- Alle Personen, deren Alter 32 ist

```
?- age(Who, 32).
Who = john ;
```

← Interpreter soll die nächste Lösung ausgeben; bei Eingabe von „a" werden alle weiteren Lösungen ausgegeben.

Zentraler Begriff in Prolog: Das Anpassen (Matchen) von Elementen der Datenbank an Variablen, nennt man Unifikation.

4. Implementierung von Regeln

Gegeben sei die Implikation

$$x \leftarrow (y_1 \wedge \dots \wedge y_n)$$

Implementierung:

$X :- Y_1, Y_2, \ldots, Y_n.$

Beispiele:

Aussage: Alle Menschen sind sterblich

Implementierung:

```
mortal(X) :- human(X).
human(socrates).
```

Abfrage:

```
?- mortal(Who).
Who = socrates
```

Komplexere Regel:

Wenn es an einem Ort regnet und friert oder schneit, sind dort die Straßen glatt.

```
glatteStrasse(X,Y) :- regnet(X,Y), friert(X,Y).
glatteStrasse(X,Y) :- schneit(X,Y).
friert(heute, München).
schneit(gestern, München).
```

Abfrage:

```
?- glatteStrasse(gestern, münchen).
Yes.
```