Fragen, Antworten, Kommentare und Hinweise

Frage: Ich habe im Internet auch den Pfeil --> in einem SWI-Prolog-Programm gesehen, was bedeutet der?

Antwort: Dies ist einer der vielen Prolog-Bereiche, die wir nicht genauer betrachten. Der Pfeil gehört zum Thema Definite Clause Grammars (DCG) mit denen die erlaubte Syntax von Texten beschrieben werden kann. Diese ist dann z. B. in Prolog-Prädikaten nutzbar. Zum etwas genaueren Einstieg, müssten Sie kontextfreie Grammatiken aus der theoretischen Informatik kennen, diese sind verwandt mit DCG und können da umgesetzt werden. Die Beispielgrammatik

Start -> A Start B | ε

A -> a

B -> b

beschreibt die Sprache mit Wörtern, die aus a-Zeichen gefolgt von genauso vielen b-Zeichen besteht. Das Epsilon soll für das leere Wort sehen.

Diese Grammatik ist in Prolog formalisierbar, wobei Nichtterminale auch mit einem kleinen Buchstaben, hier immer n beginnen müssen. Die Umsetzung in Prolog sieht wie folgt aus:

```
nStart --> nA, nStart, nB.
nStart --> [].
nA --> [a].
nB --> [b].
```

Eine Form der Überprüfung, ob ein Wort aus einem Nichtterminal abgeleitet werden kann, sieht wie folgt aus.

```
?- phrase(nStart, [a,a,b,b], []).
true .

?- phrase(nStart, [], []).
true.

?- phrase(nStart, [a,a,b,b,b], []).
false.
```

Das ist nur ein kleiner Einstieg in das Thema, wobei die oben genannten Regeln eigentlich nur syntactic sugar sind, da sie intern durch normale Prolog-Regeln ersetzt werden, die über das Prädikat listing/1 ausgegeben werden können.

```
?- listing(nStart/2).
nStart(A, B) :-
    nA(A, C),
    nStart(C, D),
    nB(D, B).
nStart(A, A).

true.

?- listing(nA/2).
```

```
nA([a|A], A).

true.
```

Eine andere Art die Regeln zu repräsentieren, ist z. B.

```
terminal(a).

terminal(b).

nichtterminal(nStart).

nichtterminal(nA).

nichtterminal(nB).

regel([nStart, nA, nStart, nB]).

regel([nStart]).

regel([nA,a]).

regel([nB,b]).
```

Statt einer Liste, könnte es auch ein zweistelliges Prädikat regel sein.

Frage: Kann man in Prolog eigentlich mit Matrizen rechnen. [ist zur Aufgabe geworden]

Antwort: Generell ja, aber wie fast immer muss entweder dafür eine passende Bibliothek gefunden werden oder eine Eigenentwicklung folgen. Da es „nur“ Listen gibt, wird ein Basistyp-Matrix zu definieren sein. Dies kann z. B. die Anzahl der Zeilen, die Anzahl der Spalten und dann eine Liste der Zeilen, die wiederum aus einer Liste bestehen, sein, z. B.

```
[3 ,2 , [[1,2],[3,4],[5,6]]]
```

Die ersten beiden Werte sind eigentlich redundant, können aber sicherstellen, dass die gewünschte Struktur eingehalten wird. Dann werden schrittweise benötigte Funktionalitäten realisiert. Soll z. B. auf ein Element an der Position x,y zugegriffen werden (at(X,Y,Wert)), ist zunächst z. B. die y-te Zeile zu bestimmen.

```
zeile([_,_,[H|_]], 0, H) :-!.
zeile([Z,_,[_|T]], Zeile, Ergebnis) :-
     Zeile > 0,
     Zeile =< Z,
     Tmp is Zeile - 1,
     zeile([Z,_,T], Tmp, Ergebnis).
```

Danach ist in einer berechneten Zeile die x-te Position zu bestimmen.

```
spalte([H|_], 0, H) :- !.
spalte([_|T], Nr, Erg) :-
```

```
      Nr > 0,
      Tmp is Nr - 1,
      spalte(T, Tmp, Erg).
```

Daraus kann dann at zusammengesetzt werden.

```
at(Matrix, X, Y, Erg) :-
      zeile(Matrix, Y, Zeile),
      spalte(Zeile, X, Erg),
      !.
```

Andere Funktionalität, wie das Addieren von Matrizen ist ebenfalls schrittweise umsetzbar, es werden einfach einzeln die Zeilen addiert.

```
matrixAdd([Z,S,[]], [Z,S,[]], [Z,S,[]]):- !.
matrixAdd([Z,S,[H|T]], [Z,S,[H2|T2]], [Z,S,[HErg|TErg]]) :-
      vektorAdd(H, H2, HErg),
      matrixAdd([Z,S,T], [Z,S,T2], [Z,S,TErg]).

vektorAdd([], [], []):- !.
vektorAdd([H|T], [H2|T2], [HErg|TErg]) :-
      HErg is H + H2,
      vektorAdd(T, T2, TErg).
```

```
?- matrixAdd([3,2,[[1,2],[3,4],[5,6]]], [3,2,[[9,8],[7,6],[5,4]]],
Erg).
Erg = [3, 2, [[10, 10], [10, 10], [10, 10]]].
```

Frage: Ich habe das mit dem Cut-Operator nicht genau verstanden, wann er genutzt werden sollte.

Antwort: Verständlich, da die Regel nur besagt, dass er eingesetzt werden kann, wenn es sicher ist, dass für den Weg bis zur Ausführung des Cut-Operators keine Alternativen mehr betrachtet werden sollen. Das ist durchaus schwammig, bedeutet praktisch aber, dass zunächst eine Lösung ohne Cut-Operator formuliert werden sollte und dass danach z. B. aus Performance-Gründen oder da es nur eine Lösung geben soll, über die Nutzung des Cut-Operators nachgedacht werden soll.

„Es ist schwer möglich allgemeine Kriterien für die sorgfältige Verwendung des „Cut“ anzugeben“ (M. Hanus, Problemlösen mit PROLOG, 2. Auflage, S.100, Teubner, Stuttgart,1987)

Relativ einfach ist das bei den genutzten Testfällen, die einfach einmal erfolgreich durchlaufen sollen, da ist der Cut-Operator am Ende passend.

Um die Problematik zu verdeutlichen hier ein Beispiel für eine falsche Generalisierung.

Falsche Annahme: Wenn ich bei Fakten angekommen bin, ist ja eine Lösung gefunden, deshalb setze ich hinter jeden Fakt einen Cut-Operator.

```
fakt1(42) :- !.
fakt1(43) :- !.
suche1(X) :- fakt1(X), X>42.
```

Die folgenden Anfragen suggerieren zunächst, dass es funktioniert. Die letzte Anfrage zeigt, dass der Cut-Operator die Suche nach einer neuen Lösung abschneidet und so kein passender Wert gefunden wird.

```
?- suche1(43).
true.

?- suche1(42).
false.

?- suche1(X).
false.
```

Mit einer kleinen Änderung, es gibt nur einen passenden Fakt, funktioniert der Ansatz aber. Wichtig ist, dass der Cut-Operator nur besagt, dass für alle Literale vor dem Cut-Operator kein Backtracking, also eine Suche nach neuen Lösungen erfolgt. Da in suche2 selbst kein Cut-Operator steht, findet eine normale Auswertung statt.

```
vorher(0).
vorher(1).
fakt2(42) :- !.
suche2(V,X) :- vorher(V), fakt2(X), Tmp is X + V, Tmp > 42.

?- suche2(A,B).
A = 1,
B = 42.
```

Mit einem falsch gesetzten Cut-Operator gibt es ein Problem, dabei kann der Cut-Operator bei fakt2(42) für das folgende Beispiel sogar weggenommen werden. Es ist wichtig, in welcher Regel oder welchem Fakt der Cut-Operator steht.

```
suche3(V,X) :- vorher(V), fakt2(X), !, Tmp is X + V, Tmp > 42.

?- suche3(1,42).
true.

?- suche3(A,B).
false.
```

Ein Cut-Operator darf auch nicht ohne Nachdenken an das Ende einer Regel gesetzt werden:

```
fakt4(42).
regel(V,X) :- vorher(V), fakt4(X), !.
suche5(V,X) :- regel(V,X), Tmp is X + V, Tmp > 42.

?- suche5(1,42).
true.

?- suche5(A,B).
false.
```