BLATT 6

DANIEL SCHMIDT & PAMELA FLEISCHMANN

Aufgabe 1. Die Methode `reachable` lässt sich wie folgt implementieren:

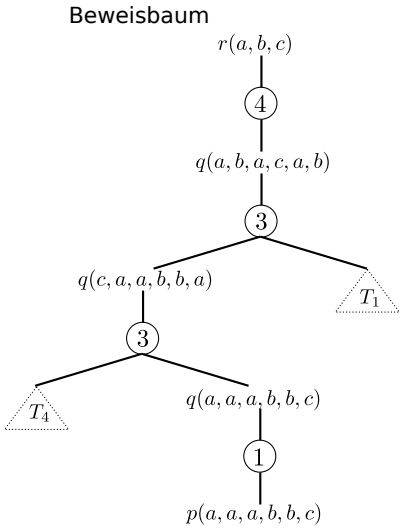
```
module serie.
export reachable(bbbf,bbbb).

reachable(G,S,[PH|PT],D) :- kante(G,S,D,PH).
reachable(G,S,[PH|PT],D) :- kante(G,S,X,PH), reachable(G,X,PT,D).

// "x"(?) macht ein Zeichen
reachable(G,S,["x"|PT],D) :- kante(G,S,D,_).
reachable(G,S,["x"|PT],D) :-  kante(G,S,X,_), reachable(G,X,PT,D).

// "y"(*) macht beliebig viele Zeichen
reachable(G,S,["y"|PT],D) :- kante(G,S,D,_).
reachable(G,S,["y"|PT],D) :-  kante(G,S,X,_), reachable(G,X,PT,D).
reachable(G,S,["y"|PT],D) :-  kante(G,S,X,_), reachable(G,X,["y"|PT],D).

end_module.
```

Ein Aufruf würde beispielsweise so aussehen:

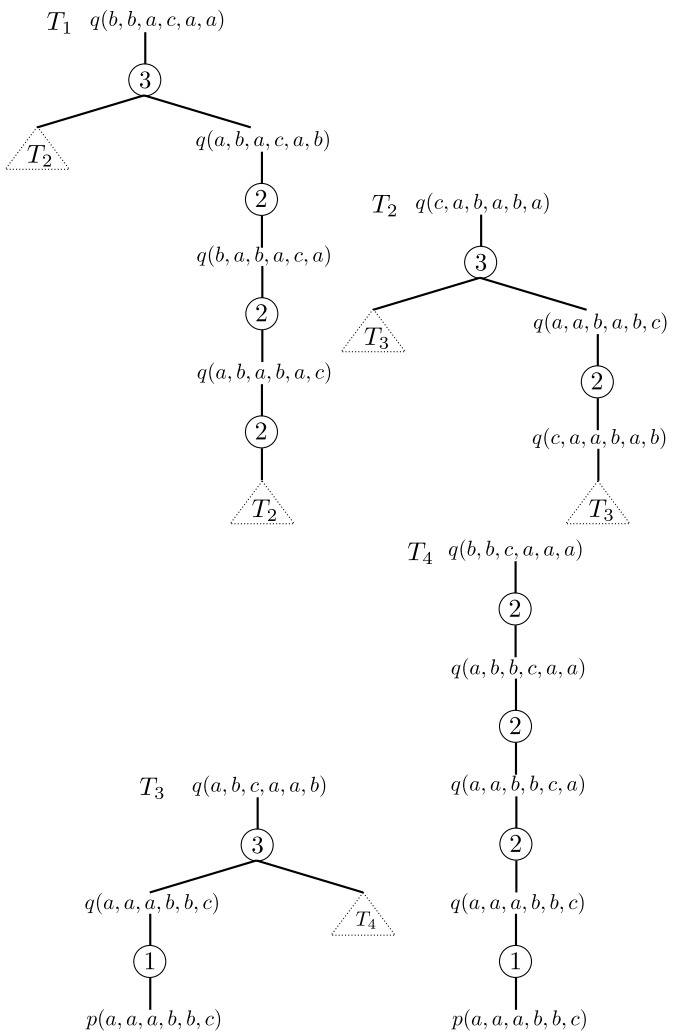
```
ready>>consult(kante.T).
ready>>consult(a1.P).
ready>>reachable(g1, "A", ["y"], Z).
CORAL::error : Not in insert mode !
ready>>?reachable(g1, "A", ["y"], Z).
Z="B".
        ... next answer ? (y/n/all)[y]
y
Z="K".
        ... next answer ? (y/n/all)[y]y
Z="C".
        ... next answer ? (y/n/all)[y]y
Z="D".
        ... next answer ? (y/n/all)[y]y
Z="E".
        ... next answer ? (y/n/all)[y]y
```

```
Z="L".
        ... next answer ? (y/n/all)[y]y
Z="F".
        ... next answer ? (y/n/all)[y]y
Z="G".
        ... next answer ? (y/n/all)[y]y
Z="J".
        ... next answer ? (y/n/all)[y]all
Z="I".
Z="H".
(Number of Answers = 11)
ready>>?reachable(g1, "A", [1,1], Z).
Z="B".
        ... next answer ? (y/n/all)[y]y
Z="K".
        ... next answer ? (y/n/all)[y]y
Z="D".
        ... next answer ? (y/n/all)[y]y
Z="E".
        ... next answer ? (y/n/all)[y]y
(Number of Answers = 4)
```

Aufgabe 2. Betrachte folgendes Datalog-Programm:
$p(a,b,c,d,e,f).p(a,a,b,b,c,c).p(a,a,a,b,b,c).p(c,c,c,b,b,a).$
$r_1 = q(U,V,W,X,Y,Z) :- p(U,V,W,X,Y,Z).$
$r_2 = q(U,V,W,X,Y,Z) :- q(Z,U,V,W,X,Y).$
$r_3 = q(X,Y,Z,U,V,W) :- q(U,V,X,W,Y,Z), q(W,Y,Z,U,V,X).$
$r_4 = r(X,Y,Z) :- q(X,Y,X,Z,X,Y).$
Sei weiter $r(a,b,c)$ ein Ziel. Dann hat ein Beweisbaum zu $r(a,b,c)$ die folgende Form



mit den Unterbäumen



Für das Beweismuster kann jedes a durch ein X, jedes b duch ein Y und jedes c durch ein Z ersetzt werden. Eine Umbennung von Variablen im Beweismuster ist nicht nötig, da im Rumpf der Regeln keine neuen Variablen eingeführt werden, sondern alle Variablen in den Rümpfen auch in den zugehörigen Köpfen vorkommen.

Aufgabe 3. Das Programm sieht so aus:

Listing 1. a3.P

```
module serie.
export is_word(f,b).

is_word(X) :- l(W,_,start), substr(X,W,R), trans(R).
trans(X) :- l(X,final,_).
trans(X) :- l(W,_,_), substr(X,W,R), trans(R).

end_module.
```

Die zugehörtige Table file die zuerst importiert werden muss so:

LISTING 2. a3.T

```
@@ 1 3
"hello" nonfinal start
"world" final nonstart
"test" nonfinal nonstart
```

Die Sprache ist definiert als: $L = hello(hello|test|world)^*world$ (wobei das "—" Symbol für oder steht).

Eine Ausgabe dieses Programms ist hier zu sehen:

LISTING 3. a3.log

```
ready>>?is_word("helloworld").
yes.
        ... next answer ? (y/n/all)[y]n
(Number of Answers = 1)
ready>>?is_word("hellotesthelloworldworld").
yes.
        ... next answer ? (y/n/all)[y]y
(Number of Answers = 1)
ready>>?is_word("testhelloworld").
(Number of Answers = 0)
ready>>?is_word("testhello").
(Number of Answers = 0)
ready>>?is_word("hellotesthelloworld").
yes.
        ... next answer ? (y/n/all)[y]y
(Number of Answers = 1)
```