Institut für Programmstrukturen und Datenorganisation
Prof. Dr.-Ing. Gregor Snelting
gregor.snelting@kit.edu
Prof. Dr. Ralf Reussner
reussner@kit.edu

Programmierparadigmen – WS 2021/22

https://pp.ipd.kit.edu/lehre/WS202122/paradigmen/uebung

Blatt 8: Cuts

Abgabe: 17.12.2021, 14:00
Besprechung: 20.12. – 21.12.2021

Reichen Sie Ihre Abgabe bis zum 17.12.2021 um 14:00 in unserer Praktomat-Instanz unter https://praktomat.cs.kit.edu/pp_2021_WS ein.

1 Tester, Generatoren, Ausführungsbäume

In der Vorlesung wurden Prädikate `odd(X)` und `even(X)` vorgestellt. Diese *testen* ob `X` gerade ist, bzw. ob `X` ungerade ist. So wird die Anfrage `?even(4)` erfüllt, `?odd(4)` hingegen nicht.

```
even(0).
even(X) :- X>0, X1 is X-1, odd(X1).

odd(1).
odd(X)  :- X>1, X1 is X-1, even(X1).
```

Listing 1: Tester

Die folgende Variante ist geeignet zum *Generieren* aller geraden bzw. aller ungeraden Zahlen. Beispielsweise gibt die Anfrage `?even(X)` bei wiederholter Neuerfüllung alle geraden Zahlen aus.

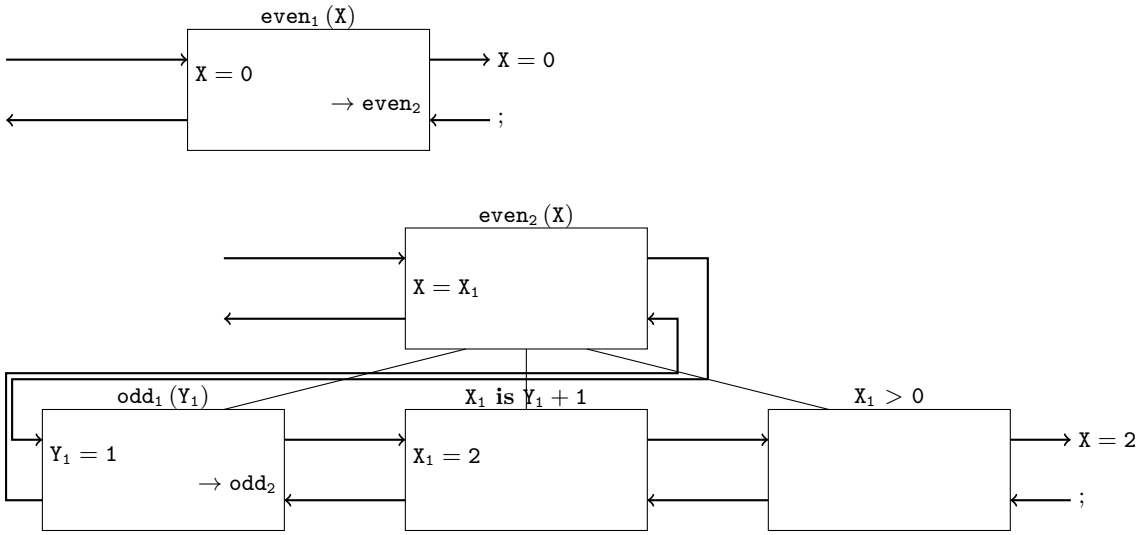
```
even(0).
even(X) :-  odd(Y), X is Y+1, X>0.

odd(1).
odd(X)  :- even(Y), X is Y+1, X>1.
```

Listing 2: Generatoren

1. Warum ist keine der beiden Varianten sowohl als Tester als auch als Generator anwendbar?
2. Was passiert, wenn bei den Generatoren die Teilziele `X>0` und `X>1` weggelassen werden?
3. Im Folgenden sind für die Anfrage `?even(X)` unter Verwendung der Generator-Regeln von oben die Ausführungsbäume zum Zeitpunkt der Ausgabe von `X=0` bzw. `X=2` dargestellt.

   Zeichnen Sie den Ausführungsbaum zum Zeitpunkt der Ausgabe von `X=4`. Welches Teilziel schlägt dabei einmal fehl?



2 Prolog, freie Variablen [basierend auf Klausuraufgabe vom WS 2014/2015]

Gegeben seien folgende Varianten eines Prädikats `del(L1,X,L2)` zum Löschen von `X` in Liste `L1`.

```
del1([],_,[]).
del1([X|T1],X,L2)      :- !, del1(T1,X,L2).
del1([Y|T1],X,[Y|T2]) :- del1(T1,X,T2).

del2([],_,[]).
del2([X|T1],X,L2)      :- del2(T1,X,L2).
del2([Y|T1],X,[Y|T2]) :- del2(T1,X,T2), not(X=Y).

del3([X|L],X,L).
del3([Y|T1],X,[Y|T2]) :- del3(T1,X,T2).
```

1. Geben Sie für jede Variante an, was diese unter der gegebenen Anfrage ausgibt. [6 Punkte]

   Anfrage: `del`$_i$`([1,2,1],X,L).`

   Mögliche Ausgaben:

   a) `X=1,L=[2]` und
      `X=2,L=[1,1]`

   b) `X=1,L=[2]`

   c) `X=1,L=[2,1]` und
      `X=2,L=[1,1]` und
      `X=1,L=[1,2]`

   d) nichts (Endlosschleife, Stacküberlauf o. Ä.)

2. Folgende Anfragen versuchen, die Rückwärtsausführung des Prädikats `del` ausnutzen. Welche Anfragen sind mit welcher Variante von $\texttt{del}_i$ erfüllbar? Warum ist das so?

   a) $\texttt{del}_i$`(L, 2, [1, 3]).`

   b) $\texttt{del}_i$`([1, 2, 3], X, [1, 3]).`

   c) $\texttt{del}_i$`([1, 2, 3, 2], X, [1, 3]).`

   d) $\texttt{del}_i$`([1, 2, 3, 2], X, [1, 2, 3]).`

   e) $\texttt{del}_i$`([1|L], 1, X).`

In Prolog können λ-Ausdrücke als Terme dargestellt werden. Wir betrachten nur Terme der Form `42` (Integer-Konstante), `x` (Variable), `abs(x,T)` (λ-Abstraktion) und `app(T,U)` (Applikation).

3. Geben Sie ein Prolog-Prädikat `fv(T,F)` an, das zu einem Lambda-Ausdruck `T` die Liste `F` der in `T` frei vorkommenden Variablen berechnet. [6 Punkte]

   Beispiel: für den Lambda-Ausdruck (λ`x`. λ`y`. `x z 17`) `u`

```
?fv(app(abs(x, abs(y, app(app(x, z), 17))), u), F).
F = [z,u];
no.
```

   Hinweis: Sie können der Einfachheit halber annehmen, dass λ-gebundene Variablen nicht anderswo im Term frei verwendet werden.
   Verwenden Sie Listenprädikate wie `append`, $\texttt{del}_i$, ... sowie Testprädikate **`integer`**`(X)` und **`atom`**`(X)`.

3 Haskell, Prolog, Listenverarbeitung [alte Klausuraufgabe, 15 Punkte]

1. Implementieren Sie eine Haskell-Funktion [8 Punkte]

```
splits :: [t] -> [([t], [t])]
```

   die alle möglichen Zerlegungen der übergebenen Liste in einen Anfangsteil und einen Endteil berechnet, so dass Anfangsteil und Endteil aneinandergehängt wieder die ursprüngliche Liste ergeben. Anfangs- und Endteil sollen jeweils in einem Tupel gespeichert und alle möglichen Tupel als Liste zurückgeliefert werden.
   Hinweis: Sie können List Comprehensions verwenden.
   Beispiel:

```
> splits [1,2,3]
[ ([],[1,2,3]), ([1],[2,3]), ([1,2],[3]), ([1,2,3],[]) ]
```

2. Implementieren Sie ein Prolog-Prädikat [7 Punkte]

```
splits(L, Res)
```

   das **bei Reerfüllung** alle möglichen Zerlegungen der übergebenen Liste `L` in einen Anfangsteil und einen Endteil generiert, so dass Anfangsteil und Endteil aneinandergehängt wieder die ursprüngliche Liste ergeben. Anfangs- und Endteil sollen jeweils als Tupel in `Res` zurückgeliefert werden.
   Beispiel:

```
? splits([1,2,3], Res).
Res = ([], [1,2,3]) ;
Res = ([1], [2,3]) ;
Res = ([1,2], [3]) ;
Res = ([1,2,3], []) ;
No
```