Datenstrukturen in Prolog

STUPS
Michael Leuschel

Einfache Terme

- Atome: c, paul, ‘X’, ‘$$_F’, ...
- Ganze Zahlen: 1, 2, ...
- Gleitkommazahlen: 1.0
- Variablen X, Y, ..., _var, ...

Eingebaute Prädikate

- number(N): N integer oder float
- var(T), nonvar(T)
- atom(T)
- atomic(T): T ist atom oder number
- simple(T): atomic oder var
- compound(T): not simple

Unifikation

- =/2, definiert wie ‘=’(X,X).
- unify_with_occurs_check/2
- \=(X,Y): wahr wenn X,Y nicht unifizierbar
- ==, \==
- ?=/2,   dif/2   wartet bis ?= wahr

Reine - Unreine Prädikate

- Kommutativität gilt für reine Prädikate:
  - Pred, fail ≡ fail, Pred
  - Pred, X=t, Y=u,... ≡ X=t, Y=u,..., Pred
- Rein: =/2, dif/2
- Unrein: ==, \=, \==, var, nonvar, print, nl, ...

Atome <-> “Strings”

- atom_codes/2, numer_codes/2, name/2
- atom_chars/2, number_chars/2

Komplizierte Terme

- (compound terms)
- Beispiele:
  - Records: date(T,M,J)
  - Peano Arithmetik: s(s(0))
  - Listen: [a,b,c]
  - Bäume tree(L, Info, R) ...

Eingebaute Prädikate

- functor/3
  - functor(f(a),F,N) ➙ F=f, N=1
  - functor(X,f,1) ➙ X=f(_)
- arg/3
  - arg(1,f(a),A) ➙ A=a
- =../2
  - f(a) =..L ➙ L=[f,a]

Weitere Prädikate

- ground(T)
- copy_term/2

Listen

- Leere Liste: Atom ‘[]’
- Nicht leere Liste: .(Head,Tail)

- .(a,[]) = Liste der Länge 1 mit a
- .(a,.(b,.(c,[]))) = Liste der Länge 3

Notationen

- [H|T] steht für .(H,T)
- [H] steht für .(H,[])
- [X,Y] steht für .(X,.(Y,[]))
- [X,Y|T] steht für .(X,.(Y,T))
- usw..

Auf wieviele Arten kann man [a,b] schreiben ?

Prädikate über Listen

- Gleichheit
- Länge (length); gleiche Länge (same_length)
- Last
- Member
- Append

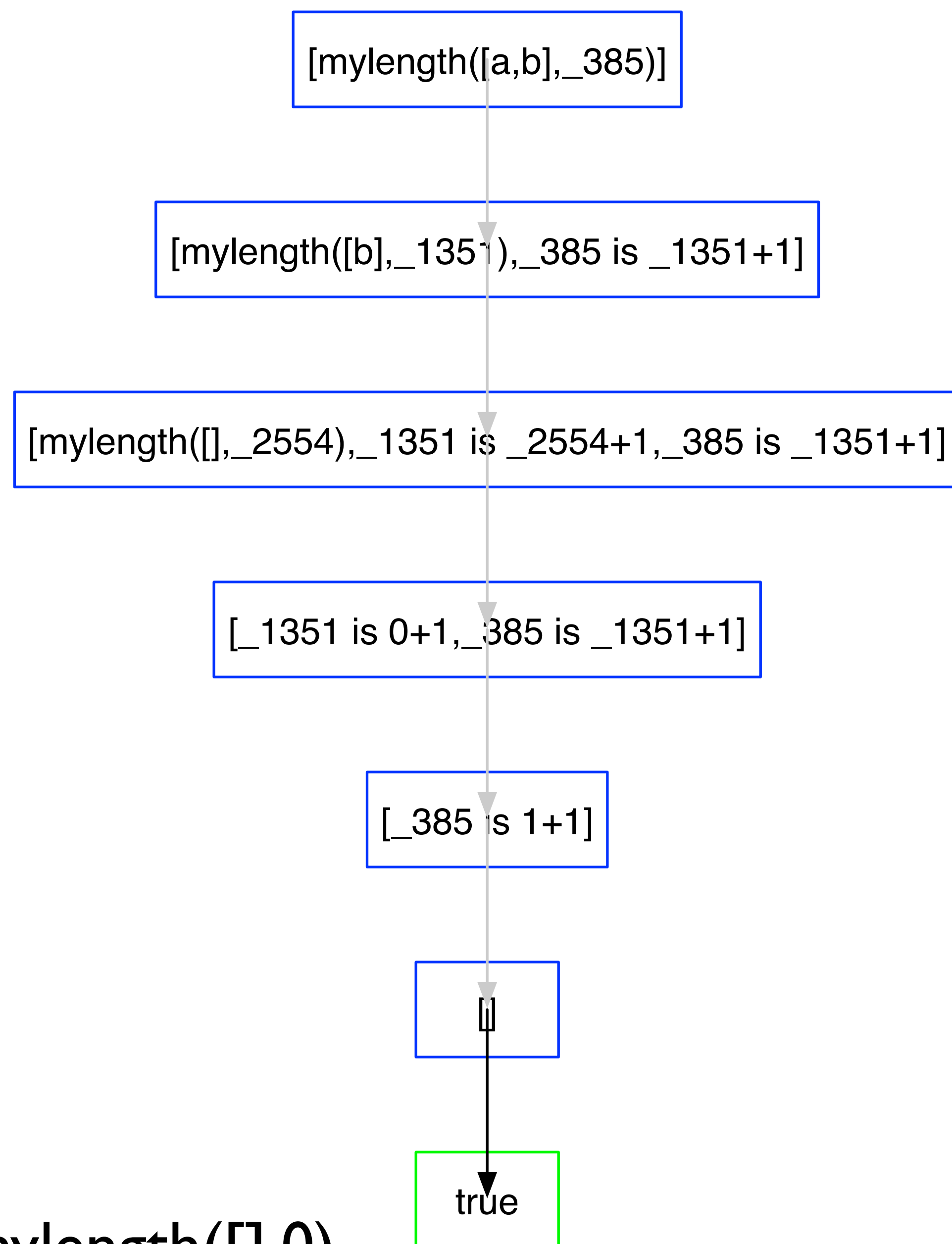
Length: 2 Versionen

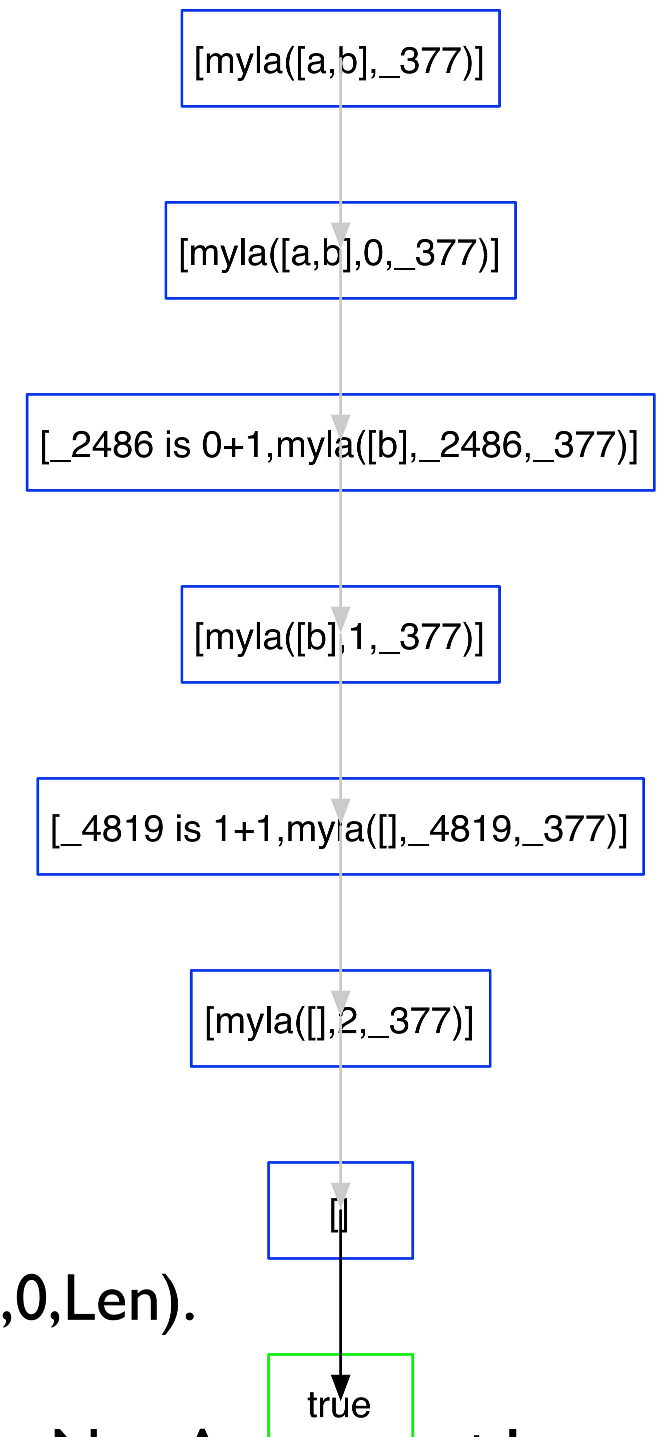
```
mylength([],0).
mylength([_|T],L) :- mylength(T,LT), L is LT+1.
```

mit Akkumulator:

```
myla(L,Len) :- myla(L,0,Len).
myla([],X,X).
myla([_|T],Acc,Res) :- NewAcc is Acc+1,
                       myla(T,NewAcc,Res)
```

Length: 2 Versionen






```
mylength([],0).
mylength([_|T],L) :- mylength(T,LT), L is LT+1.
```

```
myla(L,Len) :- myla(L,0,Len).
myla([],X,X).
myla([_|T],Acc,Res) :- NewAcc is Acc+1,
                       myla(T,NewAcc,Res).
```

Tail Recursion Optimisation

- Wenn rekursiver Aufruf
  - letzter Aufruf in der Klausel ist
  - keinen weiteren Klauseln zutreffen können
- dann wird kein neues Stack Frame angelegt

Eingebaute Prädikate

- append/3
- length/2, same_length/2
- reverse/2 (library(lists))
- nth1, perm/2, remove_dups, delete, select, sumlist, prefix, suffix, maplist, ...

Sortieren

- sort/2
- Term Order:
  - @<, @>, @=<, @>=

Binärbäume

- Kein Standard
- Es gibt aber Bibliotheken (library(avl) zB)

- Leerer Baum: leer
- Nicht leerer Baum: knoten(L,Info,R)

Prädikate für Binärbäume

- Information finden finde(Baum,Info)
- Baum nach sortierte Liste
- maximale Tiefe eines Baumes
- insert

Effizienz: Cut

- Cut (!)
  - Schneidet weitere Klauseln weg
  - Keine weiteren Lösungen links vom ! werden gesucht
- Green Cut: nur Effizienz Verbesserung
- Red Cut: Veränderung des Verhaltens

```
p(X) :- (X=1 ; X=2 ; X=3), !, print(kl1(X)),nl.
p(4).
```

Negation mit Cut

```
not(X) :- call(X),!,fail.
not(_).
```

- Ist ein “Red Cut”

Effizienz: If-Then-Else

- (Test **->** Then **;** Else)
- (Test1 **->** Then1 ; Test2 **->** Then2 ... ; **otherwise ->** Else)
- Lokaler Cut: nur eine Lösung für Test (bzw Test1, Test2,...) wird gefunden
- Etwas sauberer als !
- Manchmal schwer zu lesen

Nochmals member

- Member ohne multiple Lösungen
  - member(X,[a,b,a,a,b,a])
  - \=, !, (-> ;)

Higher-Order

- call(X)

- Map Square
  - explizit
  - mit call/2 und =../2

Findall, assert, retract

- findall(Term, Aufruf, ListenErgebnis)

- assert, asserta, assertz
- retract, retractall