

Assignment 6

Due: Friday, 23.06.2017, 15:59 via Git

For help, contact alp-staff@lists.iai.uni-bonn.de (staff only) or
alp-course@lists.iai.uni-bonn.de (staff and participants).

Start working on the exercises early enough so that you can contact your tutor in time if you have problems. Don't expect your tutor to be available at midnight or during weekends!

Submit results into the folder "assignment06/" of the git repository of your group.

For each task, submit your implemented predicate as a file named "taskN.pl". At the bottom of the file add a comment containing console output that shows **all results of a successful test run of your predicate**.

Task 1. *Implement term simplification* (25 Points)

Implement a predicate `simplify(Term,SimplifiedTerm)` with the following properties:

- Both arguments are Prolog terms that represent arithmetic expressions using the encoding introduced in the lecture (see Chapter 4, slide 40).
- `SimplifiedTerm` is obtained from `Term`
 - by replacing the sum, difference, or product of two constants by a single constant representing the result of *adding*, *subtracting* or *multiplying* the two constants, e.g.
 - $3+2$ is replaced by 5
 - $3-2$ is replaced by 1
 - $3*2$ is replaced by 6
 - by replacing the *sum of zero* with a non-constant expression with the expression, e.g. $x+0$ and $0+x$ is replaced by x
 - by replacing the *subtraction of zero* from a non-constant expression with the expression, e.g. $x-0$ is replaced by x
 - by replacing any expression at the *power of 0* with 1
 - by replacing a non-constant expression at the *power of 1* with the expression, e.g. x^1 is replaced by x
 - by replacing a non-constant expression *multiplied by 1* with the expression, e.g. x^2*1 is replaced by x^2
 - by replacing a non-constant expression *multiplied by 0* with 0, e.g. x^2*0 is replaced by 0.
- As a test, simplify the expression $2*2x^{2-1}*5^0 + x^3 * 0 - 0$ and check that the result is the term representing the expression $4x$.

- **Tip 1:** Assume that all simplification operations yield natural numbers.
- **Tip 2:** You do not need to deal with operator precedence. The Prolog parser will parse your expression properly, creating for the above example the equivalent of $(2 * 2x^{2-1} * 5^0) + (x^3 * 0) - 0$.
- **Tip 3:** See <http://www.swi-prolog.org/pldoc/man?section=functions> for the syntax of arithmetic functions that can be evaluated by the is/2 operator.

Sample solution:

```
%% simplify(+Expression, -Simplified_Expression)

%

% Toplevel from lecture slides restricted to the operations required in this task:
simplify(F+G, S) :- !, s+(F,G,S). % 0 points (copied from lecture slide 5-62)
simplify(F-G, S) :- !, s-(F,G,S). % 0 points (copied from lecture slide 5-62)
simplify(F*G, S) :- !, s*(F,G,S). % 0 points (copied from lecture slide 5-62)
simplify(exp(F,G), S) :- !, se(F,G,S). % 0 points (copied from lecture slide 5-62)
simplify(F, F). % catch all % 0 points (copied from lecture slide 5-62)

% ----- simplify sum (6 points) -----

s+(k(A),k(B),k(C)) :- !, C is A+B. % 1 point
s+(k(0),Y, Ys) :- !, simplify(Y, Ys).% 1 point
s+(Y,k(0), Ys) :- !, simplify(Y, Ys).% 1 point
s+(F,G,Result) :-
    % Simplify the input terms F and G:
    simplify(F, SF), % 0 points (copied from lecture slide 5-62)
    simplify(G, SG), % 0 points (copied from lecture slide 5-62)
    % If nothing changed stop, otherwise simplify the new sum SF+SG:
    ( SG == G, SF == F % 1 point
    -> Result = F+G % 1 point
    ; s+(SF,SG,Result) % 1 point
    ).

% ----- simplify difference (5 points) -----

s-(k(A),k(B),k(C)) :- !, C is A-B. % 1 point
s-(Y,k(0), Ys) :- !, simplify(Y, Ys).% 1 point
s-(F,G,Result) :-
    simplify(F, SF),
    simplify(G, SG),
    ( SG == G, SF == F % 1 point
    -> Result = F-G % 1 point
    ; s-(SF,SG,Result) % 1 point
    ).

% ----- simplify product (8 points) -----

s*(k(A),k(B),k(C)) :- !, C is A*B. % 1 point
s*(k(0),_, Ys) :- !, Ys=k(0). % 1 point
s*(_,k(0), Ys) :- !, Ys=k(0). % 1 point
s*(k(1),Y, Ys) :- !, simplify(Y, Ys).% 1 point
s*(Y,k(1), Ys) :- !, simplify(Y, Ys).% 1 point
s*(F,G,Result) :-
```

```
simplify(F, SF),  
simplify(G, SG),  
( SG == G, SF == F           % 1 point  
-> Result = F*G              % 1 point  
; s*(SF,SG,Result)          % 1 point  
).
```

% ----- simplify power (6 points) -----

```
se(k(A),k(B),k(C)) :- !, C is A**B . % 1 point  
se(Y,k(0), Ys) :- !, Ys = k(1). % 1 point  
se(Y,k(1), Ys) :- !, simplify(Y, Ys). % 1 point  
se(Base,Exp,Result) :-  
  simplify(Base, Sbase),  
  simplify(Exp, Sexp),  
  ( Sexp == Exp, Sbase == Base % 1 point  
  -> Result = exp(Base,Exp) % 1 point  
  ; se(Sbase,Sexp,Result) % 1 point  
  ).
```

Task 2. *Output and backtracking-driven iteration* (2 Points)

Assume there is a predicate $p/2$. Complete the following query so that *all* correct instantiations of $p(X,Y)$ are computed and printed, each one on a separate line:

```
?- p(X,Y), ... . % ← fill in the ...
```

Note: You should not need to type “;” after each result in order to get the next one!!!

Sample Solution:

```
?- p(X,Y), writeln(p(X,Y)), fail.
```

You will notice that the query always fails. To make it succeed after it has printed all possible results, you can rewrite it as follows:

```
?- p(X,Y), writeln(p(X,Y)), fail ; true.
```

A more elegant way to do the same is to introduce a new predicate, say `list_p/0`:

```
list_p:- p(X,Y), writeln(p(X,Y)), fail. % failure-based loop  
list_p. % succeed in the end
```

and then invoke

```
?- list_p.
```

Task 3. Cuts (9 Points)

Consider the following program

```
p(1).  
p(2):-!.  
p(3).
```

a) (4 Points) Write down *all* answers to each of the following queries:

- 1) ?- p(**X**). % 1 point
- 2) ?- p(**3**). % 1 point
- 3) ?- p(**X**), p(**Y**). % 1 point
- 4) ?- p(**X**), !, p(**Y**). % 1 point

b) (5 Points = 1+1+1+2) Explain for each case why the program gives that answer.

Tip: As a preparation for the written exam do it with *pen and paper only*, not by running the program! Run it and use the debugger only to verify your pen and paper answer!!!

Sample solution:

Intentionally missing. Run the program / queries to verify your answer and use the debugger to understand any difference between the real behaviour and the one you predicted!!!

Task 4. Declarative and Procedural Meaning and Cuts (3 Points)

Give the declarative and operational meaning for the predicates p, q, r. Describe the cause for differences and similarities. Recall that the declarative meaning of cut is simply "true" (it always succeeds).

```
p:- a, b.  
p:- c.  
  
q:- a, !, b.  
q:- c.  
  
r:- c.  
r:- a, !, b.
```

Sample solution:

Declarative (3 Points):

- Declaratively, p , q and r all mean the same: “(a AND b) OR c”.

Procedural (3 Points):

- $p = \text{“(a AND b) OR c”}$.
- $q = \text{“If a then b else c”}$ (see lecture slides).
- $r = \text{“c OR if a then b else false”}$. Here, the cut only affects the solution set if a has multiple definitions (e.g. two facts “a. a.”). Then the cut prevents yielding the same result twice.