Assignment 2 – with sample solutions

Due: Friday, 19.5.2017, 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).

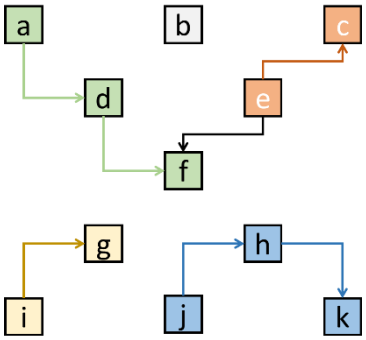
Submit results into the folder "assignment02/" of the git repository of your group.

For **task 1** submit your implemented predicate as a file named "task1.pl". At the bottom of the file add a comment containing console output that shows all results of a successful test run of path(X,Y).

For **task 2 and 3** submit your answers as a .pdf file named according to the task number.

For **task 4** submit a file "task4.pl" containing p_simple/2, q/2 and the comment required in the text of the task.

Task 1. *Path Search* (5 Points)



Write a predicate **path(Start, End)** that succeeds, if there is a path from **Start** to **End** in the graph whose arcs are represented by a set of **arc(From,To)** facts. Test your predicate on this graph:

NOTE: The graph is directed (e.g. there is no path from a to e).

Solution:

```
arc(a,d).                       % All facts together: 2 Points
arc(d,f).
arc(e,c).
arc(e,f).

arc(i,g).

arc(j,h).
arc(h,k).

path(From,To) :- arc(From,To).          % 1 Point
path(From,To) :- arc(From,X),           % 1 Point
                 path(X,To).            % 1 Point
```

Task 2. *Composition of substitutions* (3 Points)

Write down the result of each of the following substitution compositions:

a) {X←h(Z,Y)} {Y←h(Z), Z←3} ≡ { X←h(3,h(3)), Y← h(Z), Z← 3 }

b) {X←h(Z,Y)} {X← h(Z), Z← 3} ≡ {X←h(3,Y) , Z← 3} % Eliminated conflicting binding for X

c) {X←h(Z,Y)} {Y← h(Z), Z← X} ≡ { X←h(X,h(X)), % Applied second subst. to h(Z,Y)
Y← h(Z), Z← X} % Added bindings of second subst.

Note that the substitution that we obtained in c) is cyclic but still a legal composition result.
Cyclic substitutions are eliminated in the last step of the mgu algorithm (see lecture slides).

Task 3. *SLD-Resolution* (11 + 3 Points)

Write down the derivation steps that the following program performs for finding all results for the given goal.

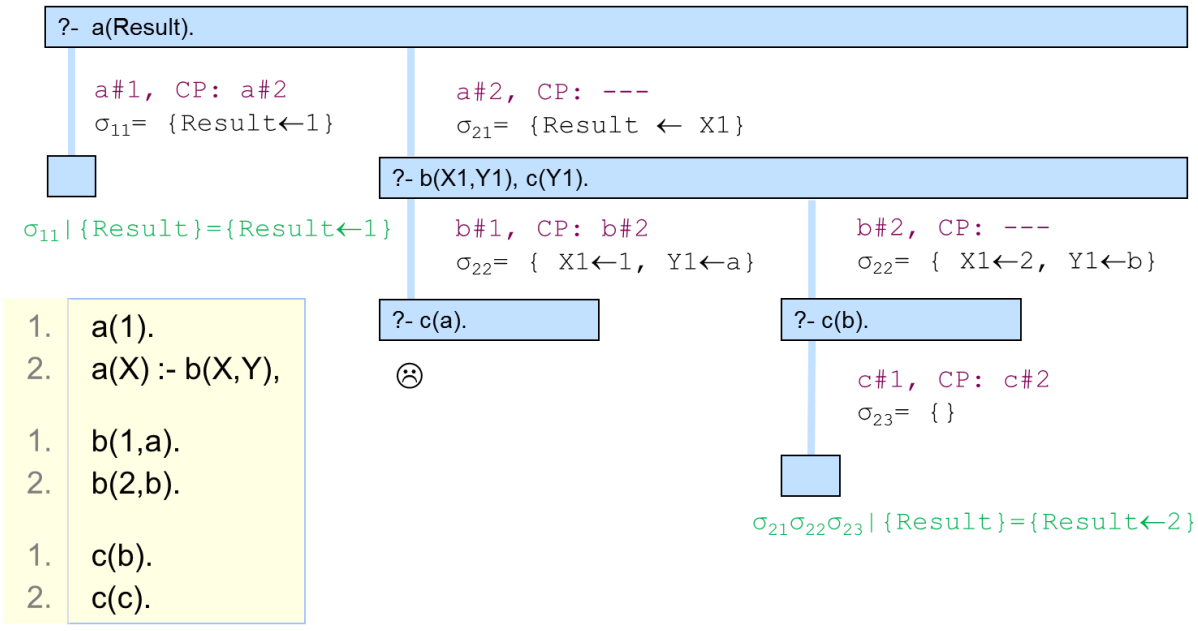
For notation, see Chapter 3 (Version from 11.05.2017), slides 44, 45, 49-52, 58-60.
Use the short-hand notation introduced on slide 58.

For each derivation step, show the number of the used clause, the computer unifier and the number of the choicepoint clause. Number each clause relative to the predicate it is part of, not relative to the entire program (e.g. the second clause of predicate b/2 is #2, not #4).

a) ?- a(Result).

a(1).
a(X) :- b(X,Y), c(Y).

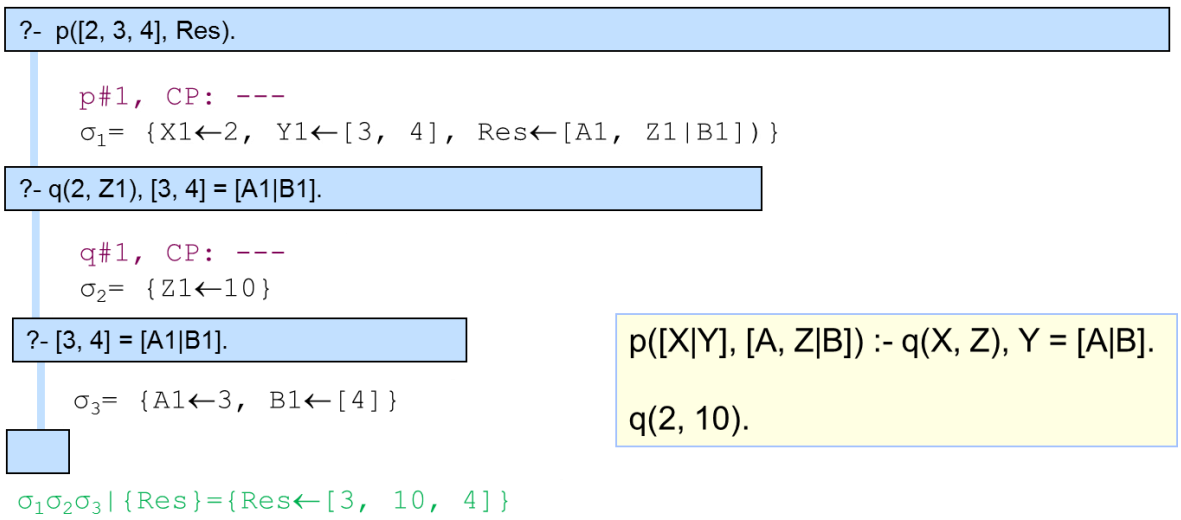
b(1, a). c(b).
b(2, b). c(c).



b) ?- p([2, 3, 4], Res).

p([X|Y], [A, Z|B]) :- q(X, Z), Y = [A|B].

q(2, 10).



Tip: Before you write down the entire derivation tree:

1. Think of the resolution steps that the interpreter will perform.
2. Write down the answer that you expect.
3. Try the query on the Prolog console. Type ";" until you got all the answers.
4. If the answers are different, think again.
5. If you cannot understand what is going on, run the query again using the debugger (working with the PDT and the debugger will be demonstrated in detail in the lecture on Monday, 15.05.2017)
6. Step through the execution and try to understand which clauses were selected, which unifications happened, which choicepoints were created, which literals in clause bodies were selected, which goals failed (and why) and the performed backtracking steps.

Task 4. *Unification and code readability* (5 Points)

Think how the definition of the predicate **p** from Task 3b) could be simplified and write a more readable version of the predicate that you call **p_simple**. Try **p_simple** with the same input as in Task 3b). Copy the result of your test from the console into a comment for **p_simple**. Add into the comment also an explanation why you think **p_simple** is equivalent to **p** on every input.

Solution:

The clause

p([X|Y], [A, Z|B]) :- q(X, Z), Y = [A|B].

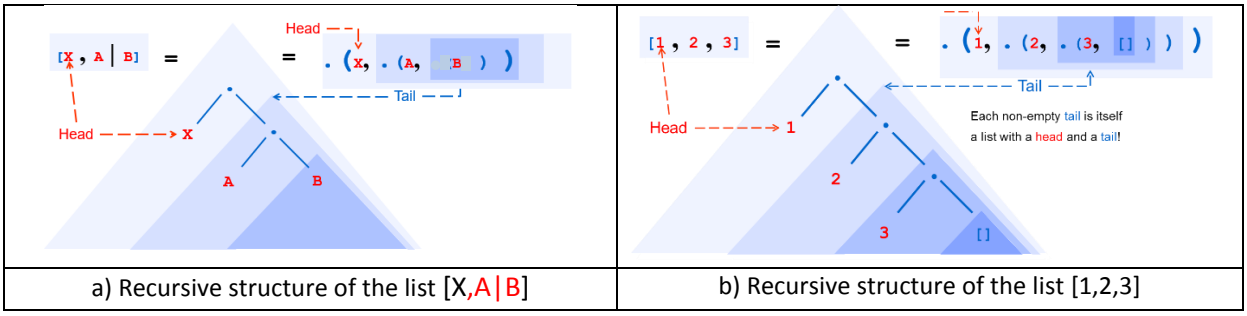
contains in its body a unification, which can be applied already before run-time, to simplify the program and save execution time. By applying the unification of Y with [A|B] to the entire clause and deleting the unification afterwards, we get (modified parts in red):

p([X|[A|B]], [A, Z|B]) :- q(X, Z).

looking at [X|[A|B]] we see that it is just an awkward notation for a list with the first element X, the second element A and the tail B, which is more readably written as [X,A|B]. This gives us

p([X,A|B], [A, Z|B]) :- q(X, Z).

Note that A is a variable that stands for the second list element whereas B is a variable that stands for the entire (yet unknown) tail of the list – 0 to arbitrarily many elements. The tree in Figure 1a) illustrates the structure of the list [X,A|B]. Figure 1b) reproduces for comparison the example from the lecture (Chapter 4, slide 15):



a) Recursive structure of the list [X,A|B]

b) Recursive structure of the list [1,2,3]