Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III
Prof. Dr. Stefan Wrobel

Advanced Logic Programming
Summer Semester 2015
Dr. Günter Kniesel

# Assignment 2

Due: Friday, 13.5.2015, 15:59:59 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 your implemented predicates as a file named “assignment02/solutions.pl” in the Git repository of your group.

Add a file “assignment02/testRuns.txt” showing the console output of a session in which you test that **each** solution works for the provided input data and some queries that represent sensible test cases.

If no input data is provided in the text of the task, create some sensible input data.
If input data is represented as facts, include them into the “solutions.pl” file and add suitable comments.

## Task 1. *JTransformer Tutorial* (6 Points)

This assignment is dedicated to getting started with practical program analysis work using Prolog and JTransformer. Start by

- installing JTransformer from sewiki.iai.uni-bonn.de/research/jtransformer/installation,
- going through the JTransformer Tutorial (http://sewiki.iai.uni-bonn.de/research/jtransformer/tutorial/stepbystep)

a) As described in the tutorial, load the JHotDraw project into your Eclipse workspace, assign it a factbase and run the query “?- classT(Id,Pkg,'DrawApplet',Members).”. Write down the values that you get for Id, Pkg and Members.
b) If you are in exercise group *N* display the *N*-th element of the Members list in the editor (via the context menu item “Show in Editor”). Copy the highlighted source code of this element as the answer to this task.
c) Show the element from b) in the Factbase Inspector and expand it so that one can see all its subelements. Submit a screenshot of this state of the Factbase Inspector.

Tip: By default, Prolog does not print deeply nested terms completely, but replaces parts beyond a certain nesting level by "...". You can change the print options, so that terms are printed to greater depth, e.g. 20, by

```
?- set_prolog_flag( toplevel_print_options, [max_depth(20),quoted(true)]).
```

## **Task 2.** *Loops* (2 Points)

Install the JTransformer Tutorial projects (via Eclipse > Examples > JTransformer). In the JT_Tutorial_Prolog/pl/utilities/shared.pl you find the implementation of a predicate "loop/4". Make sure you understand how it works (you might want to have a look at http://sewiki.iai.uni-bonn.de/research/jtransformer/api/java/pefs/3.0/java_pef_overview). Then write a detailed comment that documents the predicate (see slides 1-46 and 1-47 for an example).

## **Task 3.** *Nested Loops* (4 Points)

Write a predicate that detects nested loops: nested_loop(Outer,Inner) should succeed if

- Outer and Inner are the identities of for or while loop elements
- Outer contains Inner either immediately or deeply nested (there might be other statements within Outer that contain Inner).

Run this predicate on JHotDraw or other open source Java libraries until you find 5 occurrences of *nested* loops. Submit your Prolog program, and for each detection result the successful console output and the related Java source code.

**Tip**: Use the loop/4 predicate from the previous task and the built-in JTransformer predicate ast_ancestor(A,B), which succeeds if B is an ancestor of A (or put differently, A appears nested somewhere within B). For other generic predicates for navigating the AST structure see http://sewiki.iai.uni-bonn.de/research/jtransformer/api/meta/queries/queryapi-gen.

## **Task 4.** *Statement order in a block* (2 Points)

In Java, statements are typically contained in blocks limited by a pair of opening and closing curly braces. Such a block is represented in JTransformer by a blockT/4 fact (see http://sewiki.iai.uni-bonn.de/research/jtransformer/api/java/pefs/3.0/blockt).

Write a predicate before_in_block(+*StatementId1*, +*StatementId2*, +*BlockId*) that succeeds, if *StatementId1* comes before *StatementId2* in the statement list of the block with ID BlockID.

Hand in your program, the console output of a successful run, and the Java source code that you used for your test.

**Tip**: Use the predefined predicate nth1(?Index, ?List, ?Elem) – see the SWI-Prolog manual.

## **Task 5.** *Function terms are structured data* (11 Points)

Assume that the term representation of a binary tree is defined by the following two rules:

- The term '-' (the minus symbol) represents the empty tree.
- The term t(L,V,R) represents a tree with left subtree L, node value V, and right subtree R.

Thus t(-,1,-) is the tree that contains just the value 1 and two empty subtrees.

a) (1 Points) Draw the tree represented by the term below (values are red for easier reading):

t(t(t(-,d,-), b, t(-,e,-)) , a, t(-,c,t(t(-,g,-), f,-)) ).

b) (4 Points) Write a recursive predicate **binary_tree(Term)** that succeeds if Term is a legal binary tree according to the above definition. Use it to test whether the term from a) is a legal tree.

c) (4 Points) We say that a binary tree is symmetric if you can draw a vertical line through the root node and then the structure of the right subtree is the mirror image of the structure of the left subtree. We are only interested in the tree structure, not in the contents of the nodes. Implement a predicate symmetric/1 to check whether a binary tree is symmetric. Hint: Write first a predicate mirrors/2 to check whether one tree mirrors the structure of another tree.

d) (2 Points) Write a predicate size(Tree,Size) that succeeds whenever Size is the number of non-empty elements in Tree. Hint: Recall that arithmetic is done using the is/2 predicate. See the online SWI-Prolog manual for details.