Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III
Prof. Dr. A. B. Cremers

Advanced Logic Programming
Summer Semester 2014
Dr. Günter Kniesel

Assignment 2

Due: Monday, 5.05.2014, 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).

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 during weekends!

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. *Understanding PEF documentation* (4 Points)

Go to sewiki.iai.uni-bonn.de/research/jtransformer/api/java/pefs/3.0/java_pef_overview. Read the documentation of the program element facts (PEFs) `callT`, `getFieldT`, `identT`, `assignT`, `execT` . For an explanation of the notation see http://sewiki.iai.uni-bonn.de/research/jtransformer/api/notation. For a general introduction to the representation of Java program elements in Prolog you might want to consult http://sewiki.iai.uni-bonn.de/research/jtransformer/api/java/prologast.

Then answer the following questions:

a) Which is the argument position of the "receiver" argument in a `getFieldT`?
b) Which is the argument position of the "parent" argument in a `callT`?
c) Is there any common structure that all of the above-mentioned PEFs (`callT`, `getFieldT`, `identT`, `assignT`, `execT`) share?

Task 2. *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.

Task 3. *Loops* (6 Points)

Find out from the documentation how while and for loops are represented in JTransformer (see http://sewiki.iai.uni-bonn.de/research/jtransformer/api/java/pefs/3.0/java_pef_overview). Then write a predicate that generalizes them: loop(Id, Cond, Body) should succeed if

- Id is the identity of a (for or while) loop element
- Cond is the identity of its toplevel condition element
- Body is the list of the identity of statements inside loop block

In the case of a for loop, add its increment statement as the last statement to the body (assume for simplicity that the body contains no *return* or *continue* statements).

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

Tip: The built-in SWI-Prolog predicate `append(L1,L2,L12)` succeeds if the list L12 is the concatenation of the lists L1 and L2. For general help about built-in predicates see the online help of SWI-Prolog at http://www.swi-prolog.org/pldoc/doc_for?object=root (if you know what you are looking for just use the search field in the upper right corner).

Task 4. *Nested Loops* (4 + 6 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: The built-in JTransformer predicate ast_ancestor(A,B) 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 5. *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.