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

Advanced Logic Programming
Summer semester 2009
Dr. Günter Kniesel

Exercise Sheet 3

Due: Sunday, 24.05.2009, 23:59:59 via SVN

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

Please start working on the exercises early enough so that you can contact us in time in case of problems. Don't expect us to be available during weekend!

For the following assignments, submit

1. the implementation of each predicate,
2. the Java code on which you tested your predicates and
3. the excerpt of your Prolog Console session that documents your successful tests.

The Java program should be such that it provides examples for all the relevant cases. You may use any existing program of your own or from open source repositories. If you already submitted the program to your SVN as part of a previous assignment, you do not need to resubmit it. Just, check-in a file that specifies the repository location of the program.

See http://sewiki.iai.uni-bonn.de/research/jtransformer/ for the documentation of JTransformer's program element facts that you need.

Task 1. *Finding loops: Using disjunction for generalization* (4 Points) – 15 minutes

Implement a predicate with the signature "loop(Loop, LoopType, LoopBody)" that analyses the JTransformer representation of a Java program and succeeds whenever

- Loop is the identity of a loop,
- LoopType is the type of the loop (`for`, `while`, `foreach` or `doWhile`) and
- LoopBody is the identity of the block representing the loop's body.

Task 2. *Finding nested loops: Searching lists* (4 Points) – 15 minutes

Implement a predicate "immediately_nested_loop(OuterLoop, NestedLoop)" that succeeds whenever Nested Loop is the identity of a loop contained in the body of the loop with identity OuterLoop. Tip: The built-in predicate "member(Elem, List)" succeeds whenever Elem is an element of List.

Task 3. *Finding nested blocks: Using recursion* (6 Points) – 15 minutes

Implement a predicate "nested_block(OuterBlock, NestedBlock)" that succeeds whenever NestedBlock is the identity of a block contained directly or transitively within a statement of the block with identity OuterBlock.

Task 4. *Finding deeply nested loops: Putting it all together* (6 Points) – 15 minutes

Implement a predicate "deeply_nested_loop(OuterLoop, NestedLoop)" that succeeds whenever NestedLoop is the identity of a loop contained directly or transitively within the body of the loop with identity OuterLoop.

Bonus Task A. *Control flow* (20 Points) – 1,5 to 2 hours

Bonus tasks are optional. They are intended for advanced students to give them a challenge and also a chance to get the points needed for exam admission more quickly.

Implement a predicate "inControlFlow(Call, Statement)" that succeeds whenever Statement is (the identity of) a statement that could possibly be executed in response to the method invocation (with identity) Call. This recursively includes all statements within the control flow of any other method invocation that is in the control flow of Call.

a) How would you structure the problem? Give a "design" that specifies which helper predicates you would envisage.
b) Find out which JTransformer PEF types you need for implementing your design.
c) Implement the design.
d) Test it at least with respect to the following issues:
    a. Does it find statements at the first level of nesting?
    b. Does it find statements at deeper nesting levels?
    c. How does it deal with recursive method invocations?
    d. How does it cope with dynamic binding?
e) Reflect on what you've done. What is the critical issue (are the critical issues) of your approach?