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

AdvancedLogicProgramming
Summer Semester 2012
Dr. Günter Kniesel

Assignment3

Due: Sunday,6.05.2011, 23:59:59 via SVN

Task 1. *Understanding PEF documentation* (4 Points)

Go tosewiki.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:

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

Sample Solution:
1.a: 4
1.b: 2
1.c: Argument 1 is always the `ID`, argument 2 is the `Parent`, and argument 3 is the `EnclosingMethod`

Task 2.*JTransformerTutorial*(6 Points)

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

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

- 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.
- 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.

Sample Solution:

a)
```
?- classT(Id,Pkg,'DrawApplet',Members).
Id = 51353,
Pkg = 77311,
   Members = [51354, 77328, 77329, 77330, 77331, 77332, 77333, 77334,
   77335|...].
```

`b), c)` are straightforward

Tip: If you wonder how to see the deeply nested elements replaced above by `...` you can change the “toplevel print options”, so that terms are printed to greater depth:

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

Then run:

```
?- classT(Id,Pkg,'DrawApplet',Members).
Id = 51353,
Pkg = 77311,
Members =
[51354,77328,77329,77330,77331,77332,77333,77334,77335,77336,77337,77338,773
39,77340,77341,77342,77343,77344,51394|...].
```

Another way to see a deeply nested element of Member:

```
?- classT(Id,Pkg,'DrawApplet',Members), member(M,Members).
```

But that would take long to reach the, say, 15th element, since elements will be listed one by one. It is much more convenient to use another built-in predicate (see help/0):

```
?- classT(Id,Pkg,'DrawApplet',Members), nth1(15,Members,M).
Id = 51353,
Pkg = 77311,
Members =
[51354,77328,77329,77330,77331,77332,77333,77334,77335,77336,77337,77338,773
39,77340,77341,77342,77343,77344,51394|...],
M = 77341.
```

Task 3. *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.

Tip:

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

```
%% before_in_block(?StatementId1, ?StatementId2, ?BlockId)
%
% Succeeds if StatementId1 comes before StatementId2 in the
% statement list of the block with ID BlockID.
% Tip: Use the predefined predicate nth1(?Index, ?List, ?Elem)

before_in_block(StatementId1, StatementId2, BlockId) :-
blockT(BlockId, _, _, Elements),
nth1(Index1, Elements, StatementId1),
nth1(Index2, Elements, StatementId2),
Index1 < Index2.
```

Task 4.*Using JTransformer* (8 Points)

Write a predicate cfOrder(?*StatementId1*, ?*StatementId2*, ?*MethodId*, ?*Order*) that succeeds if *StatementId1* and*StatementId2* are the identities of statements that occur in the method with identity *MethodId*. Order isboundtotheatom

- 'before' if *StatementId1*comes before *StatementId2*in the control flow of*MethodId*
- 'after' if *StatementId1* comes after*StatementId2* in the control flow of *MethodId*
- 'none' otherwise, that is, if the two statements are on different branches of an alternative (if, case/switch).

Tips:

- Use the language-independent API for navigating through a factbase http://sewiki.iai.uni-bonn.de/research/jtransformer/api/meta/queries/queryapi-gen, in particular the ast_parent/2 and ast_ancestor/2 predicates.
- A statement S1 comes before statement S2 in the control flow if (a) the ID of S1 is before the ID of S2 in the statement list of the block that contains both or (b) the containing block of S1 is nested within a statement that comes before S2 or the statement within which S2 is nested.
- Use the predicate you wrote as a solution to the previous task.

```
cfOrder(StatementId1, StatementId2, MethodId, Order) :-
   methodT(MethodId, _,_,_,_,_,BlockId),
   before__(BlockId, StatementId1, StatementId2, Order).

% nontransitive, before
before__(BlockId, StatementId1, StatementId2, Order) :-
   before_in_block(StatementId1, StatementId2, BlockId),
   Order = before.

% nontransitive, after
before__(BlockId, StatementId1, StatementId2, Order) :-
   before_in_block(StatementId2, StatementId1, BlockId),
   Order = after.

% nontransitive, none
before__(BlockId, StatementId1, StatementId2, Order) :-
   blockT(BlockId,_,_,Elements),% For every element
   member(E,Elements),% of the block body
   ast_node_for_id(E, _, ASTNode),%get its term (not just id)
   alternative(ASTNode,Branches),% If it is an alternative
   member(StatementId1,Branches),% any pair of its disjoint
   member(StatementId2,Branches), % ... branches that are
   not(StatementId1==StatementId2),% ... not identical
   Order = none.% ... is unordered.
```

```
% transitive, before
  before__(BlockId, StatementId1, StatementId2, Order) :-
     % Whatever comes before S2
     before_in_block(StatementId1, S2, BlockId),
     ast_node_sub_trees('Java',S2,Subs), % ... is before any subelement
     nth1(_,Subs,StatementId2),           % ... of S2
     % deep(before) indicates the "transitive before" case.
     Order = deep(before).

  %%alternative( ?Term, ?List )
  %
  % The List in argument 2 contains the IDs of all alternative
  % execution paths in the statement represented by argument 1
  % In particular,
  %  - if Term is an if-statement, then List contains the IDs
  %    of its then and else part.
  %  - If Term is a switch statement, then List contains one
  %    sublist for each "case". Each sublist contains the IDs
  %    of all statements for that case.
  %
  % NOTE how we use unification in the first clause to extract
  % the relevant information from the first argument and to
  % construct the result in the second argument!

  alternative(ifT(_,_,_,_,Then,Else), [Then,Else]) .
  alternative(switchT(_,_,_,_,Cases), Statements) :-
     extract_statements(Cases,Statements).
```

NOTE:
To understand the second clause, please recall that in a PEF of the form

```
switchT(#id, #parent, #enclMethod, #expr, [#statement_1, ...])
```

the last argument contains a list of IDs of statements that includes the IDs of labels such as "case 1:", "case 2:" etc.

Let us denote

- IDs of labels by c1, c2, ...
- IDs of "break" statements (that terminate a case) by b1, b2, ... and
- IDs of other statements by s1, s2, ...

Then the statement list in the last argument of a switch/5 fact could have the following structure:

```
[ c1, s1, s2, b1,
  c2, s3, s4,
  c3, s5, s6, b3,
  c4, s7
]
```

Note that the statements executed for case 1 terminate at b1 but the statements executed for case 2 terminate at b3 because there is no break statement in b2 (see the Java Language Specification if you do not understand why).

To get sublists of statements that belong to one case we need to implement the extract_statements(Cases,Statements) predicate:

```
% Extract all statements until the next breakT into a
% sublist,  then start a new sublist:

extract_statements([],[]).

extract_statements(List,[StatementsForThisCase | More]) :-
extract_until_next_break(List,Rest,StatementsForThisCase),
extract_statements(Rest, More).



% 1.  If we find a breakT stop (return an empty list).
% 2.The ID of a caseT should not appear among the executed
% statements since the caseT is not a statement but just
%     a label that marks a place in a statement sequence.
% 3.  In any other case include the ID into the statement
%     sequence and continue

extract_until_next_break([ID|Rest], []) :-
breakT(ID,_,_,_,_).

extract_until_next_break([ID|Rest], Statements) :-
caseT(ID,_,_,_),
extract_until_next_break(Rest, Statements).

extract_until_next_break([ID|Rest], [ID|Statements]) :-
not(caseT(ID,_,_,_)),
not(breakT(ID,_,_,_,_)),
extract_until_next_break(Rest, Statements).
```