Assignment 7

Due: Friday, 30.06.2017, 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).

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 at midnight or during weekends!

Submit your implemented predicates as a file named "assignment07/solutions.pl" in the Git repository of your group. Add to each task not just the code that you implemented but also 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 JTransformer PEFs* (6 Points)

Go to https://sewiki.iai.uni-bonn.de/research/jtransformer/api/java/pefs/4.1/java_pef_overview. For the used notation see http://sewiki.iai.uni-bonn.de/research/jtransformer/api/notation, last section. For a general introduction to the representation of Java program elements in Prolog see http://sewiki.iai.uni-bonn.de/research/jtransformer/api/java/prologast and the lecture slides (Chapter 6).

Then read the documentation of the program element facts (PEFs) `compilationUnitT`, `classT`, `fieldT`, `methodT`, `callT` , `fieldAccessT`, `assignT` and answer the following questions:

a) Which is the argument position of the "members" argument in a `classT`?
b) Which is the argument position of the "parent" argument in a `fieldT`?
c) Is there any common structure of all the above-mentioned PEFs?
d) Is there anything else that `fieldAccessT`, `assignT`, `callT` have in common but not the others? Can you guess why?
e) Can you anticipate which other elements share this additional structure?

Sample Solution:

a) 5
b) 2
c) Id = 1st argument, Parent = 2nd argument

d) EnclosingMethod = 3rd argument

The EnclosingMethod argument enables direct to the enclosing method instead of having to traverse the 'parent' chain, which can take a while in deeply nested program structures. The direct access enables indexing (we will come back to this topic when we talk about efficient Prolog programming) and makes it possible to answer quickly queries of type

"Give me a method call in the body of method M.",
"Give me a field access in the body of method M.",
... etc ...

by providing the id of method M in the EnclosingMethod parameter of the respective query.

e) All elements in the body of methods i.e. Expressions and statements

Task 2. *JTransformer Tutorial* (12 Points)

This assignment is dedicated to getting started with practical program analysis. 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 first tuple of 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.

d) Select in the Editor the element that you displayed in b) and use "Copy to Clipboard":

i. Generate first a listing of all PEFs (with IDs) that represent this code
ii. Generate then a condition that will match *exactly* that code (and nothing else)
iii. Finally generate a condition that will match code with the same structure in any context.
iv. In step iii, use the option to rename the autogenerated variables to meaningful names, then paste the resulting query as the body of a new predicate that will find such pieces of code in the future. Give the predicate sensible parameters and explain briefly why you have chosen exactly these variables as parameters.

Paste each generated text (for i, ii, iii, and iv) into your solution. For the conditions and the predicate, add also the results that you got when using them as queries.

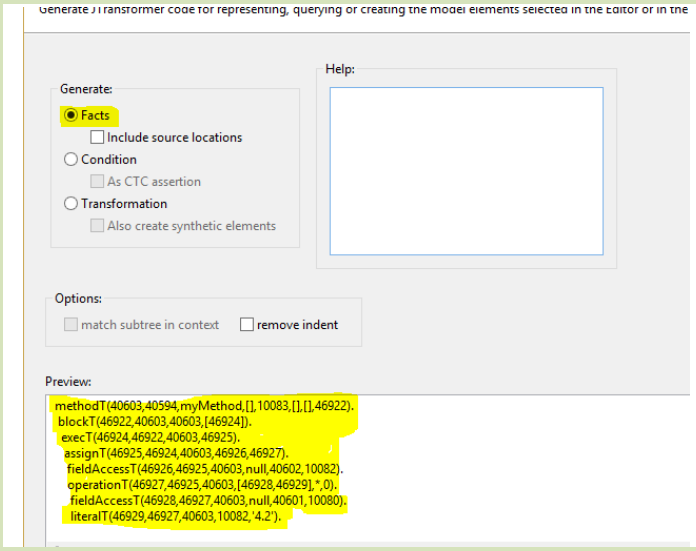
Tip: By default, Prolog does not print deeply nested terms completely, but replaces parts beyond a certain nesting level by "...". You can change the print options, so that terms are printed to greater depth, e.g. 20, by

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

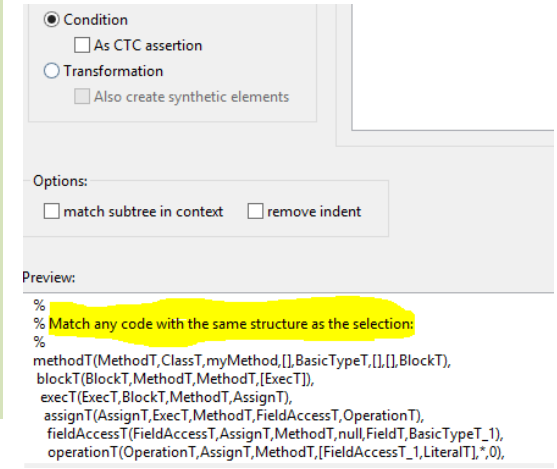
Sample Solution:

```
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 =
[51354,77328,77329,77330,77331,77332,77333,77334,77335,77336,77337,77338,77
339,77340,77341,77342,77343,77344,51394|...].
```

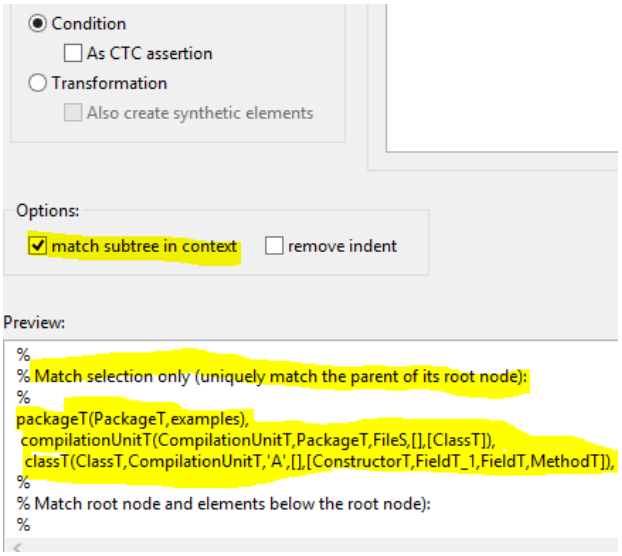
d)1.



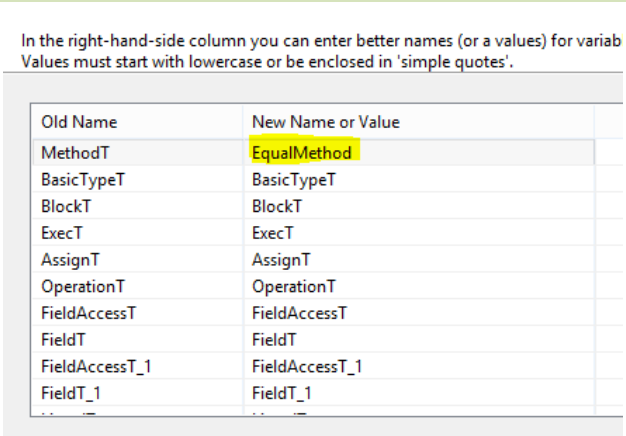
d)2.



d) 3.



d) 4.



Task 3. *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/4.1/java_pef_overview). Then write a detailed comment that documents the predicate (see slides 1-46 and 1-47 for an example).

Sample Solution:

```
%% loop(?Id,?Cond,?Body) is nondet
%
% Succeeds if Id is the identity of a for or while loop element,
% Cond is the identity of its toplevel condition element
% and Body is the List of the loop body statements
% statements of while loop and the list of body elements + updaters of
for loop.

loop(Id, Parent, EnclMethod, Cond, Body) :-
```

```
( whileT(Id, Parent, EnclMethod, Cond, BodyId),
  blockT(BodyId,_,_, Body)
;
  doWhileT(Id, Parent, EnclMethod, Cond, BodyId),
  blockT(BodyId,_,_, Body)
;
  forT(Id, _, _, _, Cond, Updaters, SyntBody),
  blockT(SyntBody,_,_,Elements),
  append(Elements,Updaters,Body)
).
```

Note that this is a slightly different version of the predicate compared to the one from the JTransformer tutorial. It includes the "Condition" and the "Body" of a loop. In the special case of the "for" loop, the statements that are executed at the end of each loop in order to change the loop counter (typically something like "i++") are syntactically not part of the loop body although conceptually they are the last statements in the body. Therefore, they are appended to the statements contained in the syntactic body of the loop (the one between the curly brackets).

Task 4. *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 any kind of loops
- Outer contains Inner either immediately or deeply nested (there might be other statements within Outer that contain Inner).

Run this predicate on JHotDraw[1] 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[2].

Tip: Use 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.

[1] JHotDraw is part of the JTransformer example projects that you get via "Eclipse > Examples > JTransformer". To run your predicate on JHotDraw, create a factbase for JHotDraw, make sure the related Prolog process is the one displayed by your Prolog Console (select it via the leftmost button at the top of the console view) and then consult the file in which you have stored your predicate definitions.

[2] To see the source code that corresponds to some ID that you get as a query result, select the ID in the Prolog console and use the "Show in editor" item of the context menu.

Sample Solution:

```
%% nested_loop(?Outer,?Inner) is nondet
%
% Succeeds if Inner and outer are IDs of loops
% and Inner is nested within Outer.

nested_loop(Outer,Inner) :-
    loop(Inner, _, _),
    ast_ancestor(Inner,Outer),
    loop(Outer, _, _).
```