CSE505 – Spring 2021

Assignment 3 – Problem 2

You may work in the same team as for A3 Problem 1

Due: ~~**Fri, April 23**~~ **Mon, April 26 (11:59 pm)**

revised due date announced on 4/15 on Piazza

Note:Assignment 4 to be given on April 23.

Unzip the directory `A3_Problem2.zip` and refer to the TINY OOPL program given in `tinyoopl.txt` – a portion of this file is shown on the next page. It gives the outline of an object-oriented program consisting of classes, fields, and methods, but *without* the bodies of methods. The constructs of a TINY OOPL program can be encoded in three relations:

`db_class(C1, C2)` – meaning, class `C1` extends class `C2`

`db_field(C, F:T)` – meaning, class `C` declares field `F` with type `T`
(where `:` is an infix binary constructor)

`db_method(C, M:T1->T2)` – class `C` declares method `M` with type `T1->T2`, where `T2` is `void` when `M` is a void method (and `->` is an infix binary constructor)

The file `database.pl` shows instances of these relations for the program in `tinyoopl.txt`, and a sample from this file reproduced on the next page. Using these relations, define the following Prolog predicates:

a. `subclass(C1,C2)`: Given `C2` as input, return in `C1` every *subclass* of `C2`, one-by-one upon backtracking. And, given `C1` as input, return in `C2` every *superclass* of `C1`, one-by-one upon backtracking. Note: the terms *superclass* and *subclass* refer not only to the immediate super/subclass, but also the super/subclasses that are obtained transitively.

b. `recursive(C)`: Given a class `C` as input, return `true`/`false` indicating whether `C` is *recursive*, i.e., whether `C` or one of its subclasses declares a field of type `C`.

c. `over_ridden(B,C,M,T)`: A method `M` of type `T` in class `C` is said to be *over-ridden* relative a class `B` if either `B` or some superclass `B2` of `B` (where `B2` is a *subclass* of `C`) also defines method `M` of type `T`.

d. `inherits(C,L)`: Given a class `C` as input, return in `L` the list of all `C2:M:T` where `M:T` is a declared (but not an *over-ridden*) method of some *superclass* class `C2` of `C`, and `T` is `M`'s declared type. The predicate should fail if `C` has no such methods.

e. `cycle(C)`: Return `true/false` indicating whether there is a *cycle* through some method of class `C`. We say there is a *cycle* through a method of class `C` if the method has a parameter with type (class) `C` or the method has a parameter some other type (class) `C2` that declares a method with a parameter of type (class) `C`; or, transitively, a class `Ck` that declares a method with a parameter of type (class) `C`.

Enter your definitions into the file `analyzer.pl`. Load into SWI Prolog the file `problem2.pl` – which includes `database.pl` and `analyzer.pl`. Proceed as follows.

```
%  prolog problem2.pl      % this might vary with platform: Mac, Linux, Windows

?- analyze.
      ... prints out banner message ...
?- subclass(c, a).
?- why(subclass(c, a)).          % 'why' provides an explanation
      …
?- cycle(a).
?- why(cycle(a)).                % 'why' provides an explanation
```

Note: The `why(G)` predicate is in the included file `explain.pl`. Sample test queries and their output are given in the file `transcript`.

WHAT TO SUBMIT: Make a directory called `A3_UBITId` if working solo or a directory called `A3_UBITId1_UBITId2` if working as a pair (give UBITId's in alphabetic order). Place in this directory your completed `A3_Problem2` directory. Compress the top-level directory and submit it using the `submit_cse505` command.

SAMPLE TINY OOPL CLASSES:

```
class a {
  int x1, y1;
  b z1;
  int f(int x1, d y1);
  void m2(d w1);
  real m3(c z1);
}

class b extends a {
  double w1;
  int x1, x2;
  c x3;
  int f(int q1, d r1);
  void m3(a z1);
}

class c extends b {
  d z1, z2;
  int f2(c q1, d r1);
  void m2(d w2);
}
```

SAMPLE DATABASE RELATIONS:

```
db_class(a, 'Object').
db_class(b, a).
db_class(c, b).


db_method(a, f, [int,d]->int).
db_method(a, m2, [d]->void).
db_method(a, m3, [c]->real).
db_method(b, m3, [a]->void).
db_method(b, f, [int,d]->int).
db_method(c, f2, [c,d]->int).
db_method(c, m2, [d]->void).


db_field(a, x1:int).
db_field(a, y1:int).
db_field(a, z1:b).
db_field(b, w1:double).
db_field(b, x1:int).
db_field(b, x2:int).
db_field(b, x3:c).
db_field(c, z1:d).
db_field(c, z2:d).
```

End of Assignment 3 Problem 2