University of Toronto
CSC410, Fall 2017

Project 2

"Your tool should be able to output a map from variables to sets of variables"

Since last time, we defined a class MYTree which can convert an expression (+ x (min 1 a))

into a list of tuples of 7 elements.

[[0, -1, '+', 1, 2, [], []], [1, 0, [], 'x', [], [], []], [2, 0, 'min', 3, 4, [], []],

[3, 2, [], 1, [], [], []], [4, 2, [], 'a', [], [], []]]

where each tuple corresponds to a node in the tree.

[idNo, parentid, op, operand1, operand2, operand3, share],

idNo is the id number for each sub-expression.

parentid is the id for parent

op is the operator such as +, -, ite, ¡=, etc

There are at most three operands, since "ite" is involved.

Each operand could be an sub-expression which is nested and long. Thus, we use an id to replace the operand in the tuple.

Algorithm:

To get the corresponding set of variables for a variable x on the left hand side, we only need to go through the the tuples in the list for the right hand side expression and test if the tuple corresponds to variable or not (i.e., the 3rd element of in the tuple is empty and the first letter of operand1 is not a digit). We have not considered the boolean constant false/true since we do not know if they should be False/Tree or false/true or just 0/1. In this way, we can get the set of variables in the expression.

After getting the pair (x, set(x)), we create a dictionary and put the pair into the dictionary.

Algorithm for naive implementation outputting

For each variable x on the left hand side, we generate the code "tempx=x". we also define a dictionary tempDic to store the pair (x, tempx), where x ia the key.

After that we generate the expression in which we replace variables also appearing on the left hand side with the corresponding temporary variables. We define a recursive function ppp(ex) in class MyTree. ppp(exp) takes a (sub) expression and converts it into a string as required by the teacher's output.

To replace a variable appearing on the left hand side with its corresponding temporary variable, we can use the following:

```
if sexp not in tempDic.keys():
                ss= str(sexp)
            else:
                ss=str(tempDic[sexp])
```

The recursive code to print the expression based on the lists from ParallelAssignments(t).raw_rhs (after parser) is as following:

```
 def ppp(self, sexp):
        ss=""
        if (isinstance(sexp, list)):
            if(len(sexp)==3):
                ss=ss+"("+str(sexp[0])+" " +self.ppp(sexp[1])+" "+self.ppp(sexp[2])+")"
        if (isinstance(sexp, list)):
            if(len(sexp)==2):
                ss=ss+"("+str(sexp[0])+" " +self.ppp(sexp[1])+")"
        if (isinstance(sexp, list)):
            if(len(sexp)==4):
                ss=ss+"("+str(sexp[0])+" " +self.ppp(sexp[1])+" "
                   +self.ppp(sexp[2])+self.ppp(sexp[3])+")"
        if (not isinstance(sexp, list)):
            if sexp not in tempDic.keys():
                ss= str(sexp)
            else:
                ss=str(tempDic[sexp])

        return ss
```

We have tested the following examples:

Example 1:

```
x,y := (+ x (min 1 a)), (* x y)
{'y': set(['y', 'x']), 'x': set(['a', 'x'])}
tempx = x
tempy = y
x = (+ tempx (min 1 a))
y = (* tempx tempy)
```

Example 2:

```
x,y,z := (+ (+ 12 (+ y y34)) (- 12 x3)),  (- z112 (- 12 z)), (* x123  (+ x 12))
{'y': set(['z112', 'z']), 'x': set(['y', 'x3', 'y34']), 'z': set(['x', 'x123'])}
tempx = x
tempy = y
tempz = z
x = (+ (+ 12 (+ tempy y34)) (- 12 x3))
y = (- z112 (- 12 tempz))
z = (* x123 (+ tempx 12))
```