CSSE4630 Week 4 Lab: Implementing Simple Dataflow Analyses

Mark Utting

Version 1.1

1 Introduction

In this workshop, we will implement the missing parts of some simple dataflow analyses in the Scala implementation of TIP:

- Simple Sign Analysis
- Powerset Lattice

If you did not set up Scala, SBT and TIP on your computer last week, please do this now — see the Week 3 workshop instructions.

To check if you have it correctly installed, you should be able to run the 'tip' script at the top level of the TIP repository. Right click on it in IntelliJ and do `Run tip`. This should show a help message in the console output. We want to run this script with parameters, corresponding to the following command line:


```
tip -run examples/constants2.tip
```

Note: You can also do this from within IntelliJ: right-click on the 'tip' script and use the `Edit tip...` menu item to edit the run configuration to add the additional parameters — do not include the initial 'tip' script name.

This `constants2.tip` program will prompt you to input an integer, and will then print 12.

```
main() {
    var x,y,z;
    x=27;
    y=input;
    z=2*x+y;
    if( 0>x ) {
        y=z-3;
    } else {
        y=12;
    }
    return y;
}
```

2 Implement Simple Sign Analysis

The simple sign analysis in `src/tip/analyses/SimpleSignAnalysis.scala` is not fully implemented. To see this, run the 'tip' script with the `-sign` option, to turn on sign analysis:

```
tip -sign -run examples/constants2.tip
```

You should see an error about unimplemented methods, like this:

```
[error] scala.NotImplementedError: an implementation is missing
[error]         at scala.Predef$.$qmark$qmark$qmark(Predef.scala:288)
[error]         at tip.analysis.SimpleSignAnalysis.localTransfer(SimpleSignAnalysis.scala:89)
. . .
```

Fix this problem by completing the implementation in `SimpleSignAnalysis.scala`.

You 'just' need to implement the missing code (the triple question marks) inside the `localTransfer` method. That is, decide what happens for variable declarations, and for assignment statements.

Recall that the sign analysis rule for variable declarations `var X1,...,Xn` is:

$$[[v]] = JOIN(v)[X1 \mapsto \bot; \ldots; Xn \mapsto \bot]$$

And the sign analysis rule for assignment $X = E$ is:

$$[[v]] = JOIN(v)[X \mapsto eval(JOIN(v), E)]$$

The Scala `localTransfer` function is just implementing the transfer function for each node, so the $JOIN(v)$ has already been done and the input parameter `s` is the result of that join. So your code must just implement the updates after the join — a single update for the assignment, and a multiple update on all the variables `X1,...,Xn` for the variable declarations.

Note: you can update a Scala map `m` with a single `key` and `value` entry by writing:

```
m + (key -> value)
```


or update multiple entries in the map by using the `++` operator with a list of `(key,value)` pairs.

To see if you have finished the Sign Analysis correctly:

1. Recompile the SimpleSignAnalysis.scala file. No errors is the first good sign.
2. Run tip as follows: `tip -sign examples/constants2.tip`
3. If that works without errors, it will create an output file `out/constants2.tip__sign.dot`. You can view this file with a text editor to see the abstract Sign value associated with each variable at each control-flow node.
4. But even better is to view this output file graphically — *.dot files are graph files that can be viewed either by installing the GraphViz tools from `https://graphviz.org`, or by using a web-based GraphViz viewer such as `http://www.webgraphviz.com`.

You should see an output graph like the one shown in Fig. 1. Note that the two colon-separated numbers after each statement and variable name are the line number and the column number where that statement appeared in the source program, or where the variable was declared.

3 Powerset Lattice

Implement the missing parts of the PowersetLattice in **src/lattices/GenericLattices.scala**.

After you have implemented the missing methods, test your lattice operations using the Scala REPL (Read, Evaluate, Print Loop). See **Tools / Scala REPL...**.

You can try experiments like the following (input lines start with a `scala>` prompt):

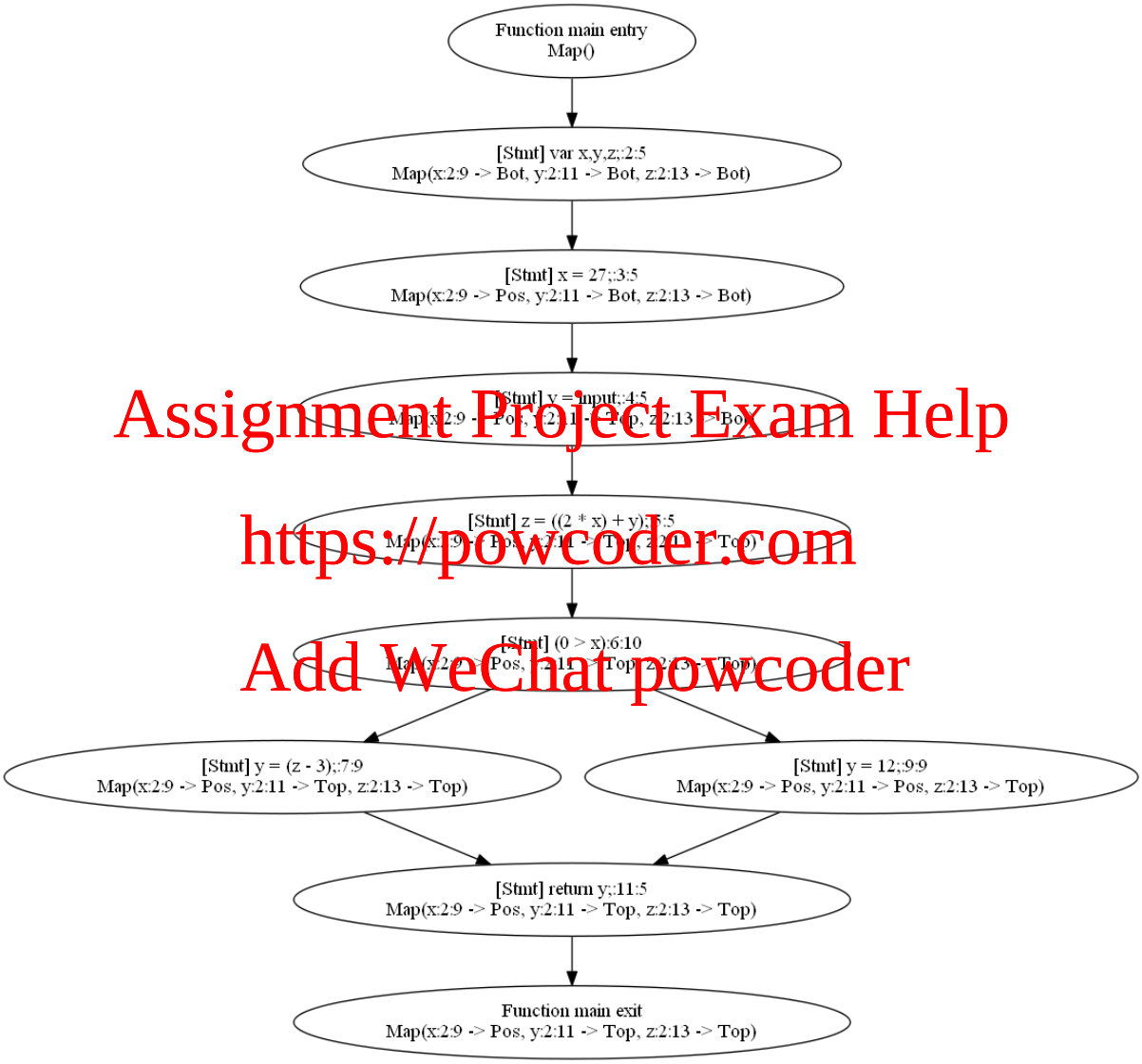

Figure 1: Results of Simple Sign Analysis for TIP/examples/constants2.tip

```
scala> var p = new tip.lattices.PowersetLattice[Int](i => 0 <= i && i < 8)
p: tip.lattices.PowersetLattice[Int] = tip.lattices.PowersetLattice@45e11627

scala> p.bottom
res0: scala.collection.immutable.Set[Int] = Set()

scala> p.top
scala.NotImplementedError: an implementation is missing
  at scala.Predef$.$qmark$qmark$qmark(Predef.scala:288)
  at tip.lattices.Lattice.top(GenericLattices.scala:35)
  at tip.lattices.Lattice.top$(GenericLattices.scala:35)
  at tip.lattices.PowersetLattice.top(GenericLattices.scala:148)
  ... 28 elided

scala> p.lub(Set(1,3,5), Set(1,2,3))
res2: scala.collection.immutable.Set[Int] = Set(1, 3, 5, 2)

scala> p.leq(p.bottom, Set(1,2))
res3: Boolean = true

scala> p.leq(Set(1,3), Set(1,2))
res4: Boolean = false
```


Note that the lattices do not implement 'top' by default. Can you implement it in your PowersetLattice?

4 Help!

If you get stuck with Scala, Google it, or use the docs:

- ScalaBook Prelude: `https://docs.scala-lang.org/overviews/scala-book/prelude-taste-of-scala.html`
- CheatSheet: `https://docs.scala-lang.org/cheatsheets/index.html`
- Main Docs page: `https://docs.scala-lang.org`

Note: if you are familiar with Python list comprehensions, the rough equivalent in Scala is to use a 'yield' expression with a 'for' loop, in order to produce a list of values:

```
scala> for (i <- 0 until 8) yield i*i
res0: scala.collection.immutable.IndexedSeq[Int] = Vector(0, 1, 4, 9, 16, 25, 36, 49)

scala> for (i <- 0 until 3) yield (i, i*i)
res1: scala.collection.immutable.IndexedSeq[(Int, Int)] = Vector((0,0), (1,1), (2,4))
```

You can experiment with these kinds of Scala expressions using the IntelliJ **Tools / Scala REPL...** menu.