Assignment 3: Dataflow Analysis

Milan Lagae

Institution/University Name:	Vrije Universiteit Brussel
Faculty:	Sciences and Bioengineering Sciences
Course:	Software Quality Analysis

Contents

1 Intro

2 Discussion Point 1

2.1 Implementation

2.1.1 Asserts

```
case ABinaryOp(GreatThan, id: AIdentifier, ANumber(i, _), _) => {
  val xDecl = id.declaration
  // Get the interval for the declaration
  val old = s(xDecl)
  // Create the new interval by applying (zero is ignored?)
  val newInterval = widenInterval(old, (i, 0))
  // Update with the new interval
  s.updated(xDecl, newInterval)
}
```

Listing 1: Assert - Version 1

```
case ABinaryOp(GreatThan, id: AIdentifier, ANumber(i, _), _) => {
  val xDecl = id.declaration
  // Get the interval for the declaration
  val old = s(xDecl)
  // Create the new interval by applying (zero is ignored?)
  val newInterval = widenInterval(old, (i, 0))
  // Update with the new interval
  s.updated(xDecl, newInterval)
}
```

Listing 2: Assert - Version 2

2.1.2 Widen Interval

```
case ((l1, h1), (l2, h2)) => {
  IntervalLattice.intersect((l1, h1), (l2, IntervalLattice.PInf))
}
```

Listing 3: widenInterval

2.1.3 Assignment(s)

```
// var declarations
// ⟨vi⟩= JOIN(vi)
case varr: AVarStmt => {
  varr.declIds.foldLeft(s) { (state, decl) =>
    state.updated(decl, valuelattice.top)
  }
}
```

Listing 4: Declarations

```
// assignments
// ⟨vi⟩= ⟨x=E⟩= JOIN(vi)[x ↦ eval(JOIN(vi), E)]
case AAssignStmt(id: AIdentifier, right, _) => {
  val interval = eval(right, s)
  s.updated(id, interval)
}
```

Listing 5: Declaration

2.2 Results

2.3 Analysis Precision

Question(s): What would be the most precise result? Why does the analysis lose precision on this program?

TODO: Most precise result

3 Discussion Point 2

3.1 Implementation

3.1.1 Context

```
// MOD-DP2
def makeLoopContext(c: CallStringContext, n: CfgNode, x:
statelattice.Element): CallStringContext = {
  // Add node to call string context, while maintaining limit on
context length
  CallStringContext((n :: c.cs).slice(0, maxCallStringLength))
}
```

Listing 6: Loop Context

3.1.2 Unrolling

```
//// Discussion Point 2: COMPLETE HERE
// Thus, to determine the starts and ends of loops you must use the
cfg.dominators function.
case m: CfgStmtNode if loophead(n) => {
  val node = m.data
  println("Current Node: " + node.toString)
  println("Successor Node: " + (m.succ intersect
dominators(m)).head.data)

  val loopStart = (m.succ intersect dominators(m)).head
  val newContext = makeLoopContext(currentContext, loopStart,
s)
  println("Context: " + newContext)
  println("S: " + s)

  propagate(s, (newContext, m))

  s
}
```

Listing 7: Loop Unrolling

3.2 Results

4 Discussion Point 3

4.1 Context

Question: Which variables would you include in the context for functional loop unrolling?

Since the bases of functional sensitivity is on the abstract state, it would atleast start of with the variable(s) defined in the predicate of the while loop. The more variables added to the context that are defined/used inside of the loop the more precision is gained. Increasing the size of the state to be stored in the context, comes with the drawback that performance might be reduced.

Continuining from the context with atleast variable i, variable x, defined in the loop may also be added.

4.2 Question 2

Question: Write a TIP program where functional loop unrolling improves precision compared to callstring loop unrolling, and explain the difference.

TODO: Add

4.2.1 Program

```
x = 1;
y = input

while (i > 0) {
  assert i > 0;
  if (i % 2 == 0) {
    x = x + 1;
  } else {
    if (x > 0) {
      assert x > 0;
      x = x - 1;
    }
  }
}

return x;
```

Listing 8: Example program - functional loop unrolling.

TODO: Add

4.2.2 Difference

TODO: Add

4.3 Finite

Question: Does interval analysis with functional loop unrolling terminate for every program? Explain why or why not (give an example).

Applying the practice of loop unrolling to functional sensitivity does not change the fact that for some given programs the analysis will **not** terminate. An example for sucha program can be seen in Listing 9.

```
x = 0;

// First iteration
x = x + 1;

// While iteration
while (true) {
  x = x + 1;
}
```

Listing 9: Example program - functional loop unrolling.

As with functional sensitivity for each abstract state of the program, in this the `while` loop a new context is generated [1], [2]. Unrolling the first iteration of the loop as displayed in the above program, does not terminate for the given program, since the size of the state (on which functional sensitivity based itself) is not finite in this case. Therefore when considering functional sensitivity, the chosen state is to be considered carefully [1].

Bibliography

[1] [Online]. Available: https://cs.au.dk/~amoeller/spa/7-interprocedural-analysis.pdf

[2] [Online]. Available: https://dl.acm.org/doi/fullHtml/10.1145/3230624