CS243 Homework 7 Solutions

Winter 2017

Problem 1. Pointer Analysis (15 points)

1. `hP` tuples for context insensitive analysis:
   `hP(h1, a, h2)`
   `hP(h1, a, h3)`
   `hP(h2, a, h2)`
   `hP(h2, a, h3)`
   `hP(h3, a, h1)`
   `hP(h2, b, h1)`
   `hP(h2, b, h2)`
   `hP(h3, b, h1)`
   `hP(h3, b, h2)`

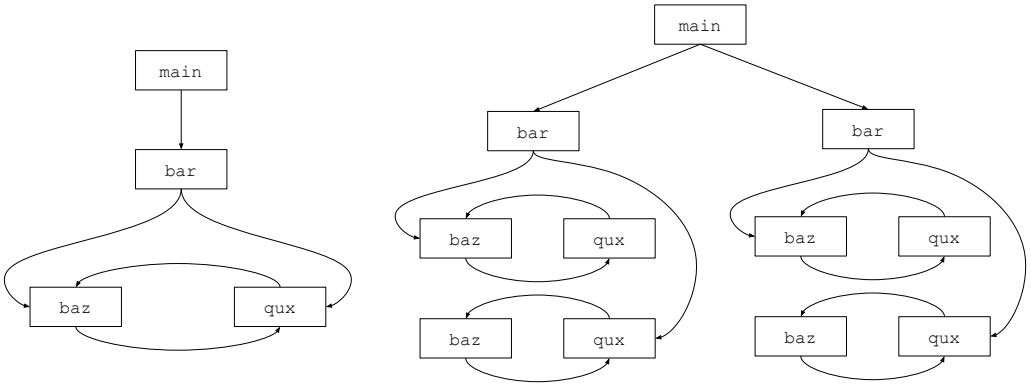
   From the declarations of `x, y, z`, you have `vP(x, h1)`, `vP(y, h2)` and `vP(z, h3)`. For context insensitive analysis, for `bar`, you have, `assign(p, x)`, `assign(q, y)`, `assign(p, y)` and `assign(q, z)` relations, that is, `p` could be `x` or `y` and `q` could be `y` or `z`. A similar context insensitive analysis for `baz` and `qux` gives you more relations, with the result that `x, p, c, m` could point to `h1` or `h2`, and `y, q, d, n` could point to `h2` or `h3` (more `vP` tuples). Finally, taking a join with relations `store(z, a, x)`, inferred from `main`, `store(c, a, d)`, inferred from `baz`, and `store(n, b, m)`, inferred from `qux`, you can derive the complete set of `hP` relations above.

   A common mistake was that results for context sensitive analysis were provided here. Note that for context insensitive analysis, for `bar`, `p` could be `x` or `y` and `q` could be `y` or `z`. You do not have any context information, and so this results in more relations than context sensitive analysis.

2. Call-graphs, context insensitive one on the left, and context sensitive one on the right:

3. Copies of procedures
   `bar`: 2 `baz`: 4 `qux`: 4

4. `hP` tuples for context sensitive analysis:
   `hP(h1, a, h2)`
   `hP(h2, a, h3)`
   `hP(h3, a, h1)`
   `hP(h2, b, h1)`
   `hP(h3, b, h2)`



Here, you run the analysis separately for each clone. For the first `bar`, `p` can be `x` only, pointing to `h1`, and `q` can be `y` only, pointing to `h2`. Further running the analysis, `c` and `m` can point to `h1` and `d` and `n` can point to `h2`. This gives the 2 relations `hP(h1, a, h2)` and `hP(h2, b, h1)`. The relations `hP(h2, a, h3)` and `hP(h3, b, h2)` can be derived similarly for the second `bar`. `hP(h3, a, h1)` is from the `main` program.

Problem 2. SMT (15 points)

View online at: http://rise4fun.com/Z3/iov5?menu=0

```
; SSA Form:
; phi1 = 0 <= x && x < N
; phi2 = x <= N/2
; x1 = x + 1
; x2 = phi2 ? x1 : x
; v = data[x2]
; phi3 = 0 <= v && v < N
; z1 = data[v]
; z2 = 0
; z3 = phi3 ? z1 : z2
; data[z3]

; Declare arguments
(declare-fun x () (_ BitVec 32))
(declare-fun data () (Array (_ BitVec 32) (_ BitVec 32)))
(declare-fun N () (_ BitVec 32))

; Declare the rest of the SSA variables
(declare-fun x1 () (_ BitVec 32))
(declare-fun x2 () (_ BitVec 32))
(declare-fun v () (_ BitVec 32))
(declare-fun z1 () (_ BitVec 32))
(declare-fun z2 () (_ BitVec 32))
(declare-fun z3 () (_ BitVec 32)) ; meta

; Declare the path variables
(declare-fun phi1 () Bool)
(declare-fun phi2 () Bool)
(declare-fun phi3 () Bool)

; For convenience
(declare-fun zero () (_ BitVec 32))
(declare-fun one () (_ BitVec 32))
(assert (= zero #x00000000))
(assert (= one #x00000001))

; Main translation of the program into SMT. Here we are
; only concerned with the dataflow, not control flow
(assert (= phi1 (and (bvsle zero x) (bvslt x N))))
(assert (= phi2 (bvsle x (bvashr N one))))
(assert (= x1 (bvadd x one)))
(assert (= x2 (ite phi2 x1 x)))
(assert (= v (select data x2)))
```

```
(assert (= phi3 (and (bvsle zero v) (bvslt v N))))
(assert (= z1 (select data v)))
(assert (= z2 zero))
(assert (= z3 (ite phi3 z1 z2)))

; Check for the first access on line 7
(push)
(assert phi1) ; We need to take the first branch
(assert (not (and (bvsle zero x2) (bvslt x2 N))))
(check-sat)
(get-model)
; The model tells us that x = 0, N = 1, data = {2,0}. This is
; a real bug because x2 will be 1, which is not less than N.
; Note that the rest of the model parameters aren't relevant.
(pop)

; Check for the second access on line 9
(push)
(assert phi1)
(assert phi3)
(assert (not (and (bvsle zero v) (bvslt v N))))
(check-sat)
; Unsat, which means this can't crash. This is pretty obvious
; because the access is protected by a check.
(pop)

; Check for the third access on line 13
(push)
(assert phi1)
(assert (not (and (bvsle zero z3) (bvslt z3 N))))
(check-sat)
(get-model)
; The interpretation of this model is more complex. Because this
; access is after the line 7 access, it is possible this code is
; never executed when the program has already crashed at line 7.
; That is what this model says: we have the same x=0, N=1 problem
; as before. This model causes a crash, just not a crash at the point
; in the program we were expecting. The translation is sound in the
; sense that any found errors will crash the program, but these errors
; may not be located correctly. Like many of the analyzes we have
; seen this quarter, this imprecision can be good or bad depending on
; the indended application. See below for how to fix this.
(pop)


; Extra (We did not expect you to do this on the homework)
; Check for the third access on line 13 correctly
```

```
(push)
(assert phi1)
; Additionally require the access at line 7 to be in bounds.
; We don't need to check line 9 because it is always valid.
(assert (and (bvsle zero x2) (bvslt x2 N)))
(assert (not (and (bvsle zero z3) (bvslt z3 N))))
(check-sat)
(get-model)
; The model now is x = 0, N = 129, data = {0x80002, 0, 0x80002, 0x80002...}
; The program sets v = data[1], thus setting z3 = 0x80002 which is obviously
; out of bounds of N = 129
(pop)
```