CS 243 - Section 3

Winter 2021

January 29, 2021

1 Taint Analysis

Data from untrusted sources could cause security vulnerabilities in programs. For example, in C, a hacker can potentially take over the control of a program by passing well crafted strings into a format function (e.g. `printf()`).

Consider a simplified language whose variables can only take string values and has the following kinds of statements:

```
ASSIGNMENT: a = <constant string>
COPY: a = b
CONCATENATION: a = b + c
INPUT: a = read()
PRINT: print(a)
CONDITIONAL BRANCH: if (a) goto L
```

A variable is tainted if it is defined by an `INPUT` statement, or a `CONCATENATION` or `COPY` of other tainted variables. This problem has two parts:

(a) Use data flow analysis to issue warnings whenever a `PRINT` statement may write out a tainted variable.

(b) Indicate for each warning the `INPUT` statements that may be responsible for the warning. For example, for the following code:

```
L1: a = read ()
L2: c = read ()
L3: if (a) goto L5
L4: a = c
L5: b = a + "str"
L6: print (c)
L7: print (b)
```

Warnings should be issued on both L6 and L7.

`L6 warning: input statements that may be responsible:` {`L2`}

`L7 warning: input statements that may be responsible:` {`L1, L2`}

<table>
<tr><td>Direction of your analysis
(forward/backward)</td><td>
Backwards.</td></tr>
<tr><td>Lattice elements and meaning</td><td>For each <code>PRINT</code> statement: At the given program point, the set of variables that influenced this <code>PRINT</code> statement.</td></tr>
<tr><td>Meet operator or lattice diagram</td><td>For a single <code>PRINT</code> statement, the diagram is the same as live variable analysis (Union on set of variables). The semi-lattice diagram for a program is the product of the diagrams of all <code>PRINT</code> statements.</td></tr>
<tr><td>Is there a top element?
If yes, what is it?</td><td>
Empty set.</td></tr>
<tr><td>Is there a bottom element?
If yes, what is it?</td><td>
Universal set.</td></tr>
<tr><td>Transfer function of a basic block</td><td>If not explicitly stated, in[B] = out[B]. We perform the following logic for each <code>PRINT</code> statement in the program.
<table>
<tr><td><code>a = <str></code></td><td>in[B] = a ∈ out[B] → out[B] - {a}</td></tr>
<tr><td><code>a = b</code></td><td>in[B] = a ∈ out[B] → (out[B] - {a}) ∪ {b}</td></tr>
<tr><td><code>a = b + c</code></td><td>in[B] = a ∈ out[B] → (out[B] - {a}) ∪ {b, c}</td></tr>
<tr><td><code>a = read()</code></td><td>in[B] = a ∈ out[B] → out[B] - {a}</td></tr>
<tr><td><code>print(a)</code></td><td>in[B] = out[B] ∪ {a}</td></tr>
<tr><td>other</td><td>in[B] = out[B]</td></tr>
</table></td></tr>
<tr><td>Boundary condition initialization</td><td>IN[EXIT] = ∅</td></tr>
<tr><td>Interior points initialization</td><td>∅</td></tr>
</table>

State explicitly how you generate the warnings and the causes of the warnings.

After running the data flow analysis, we go through the program one more time. For each `INPUT` block B, if the variable that got assigned by the `INPUT` is in OUT[B] for a `PRINT` statement P, we put the line number of the `INPUT` in P's set. We then issue warnings for each `PRINT` if there are any `INPUT` line numbers in the set. NOTE: There's another forward data flow analysis solution that can solve this problem, which is also acceptable.

2 Monotonicity and Fixed Points

(a) It is a common misconception that the definition of monotonicity is $x \leq f(x)$. Show a monotonic function (including its semi-lattice and meet operator) that does not obey $x \leq f(x)$.

Consider the semi-lattice of natural numbers ($n \geq 1$), with the min meet operator. Identity, associativity, and commutativity can easily be checked. Note that $\min(1, n) = 1$ for any n so $\perp = 1$. We can see that our partial order is the $\leq$ operator over the natural numbers.

Now let our transfer function be $f(x) = \max(x - 1, 1)$. The true definition of monotonicity is $x \leq y \implies f(x) \leq f(y)$. We can easily see f is monotonic, since $x \leq y \implies x - 1 \leq y - 1$ (even if x or y is 1, it'll stay at 1 and we can't go lower than 1 in our semi-lattice anyways). However, $x \leq x - 1$ is always false.

(b) Let us focus on the product semi-lattice of all basic block entry points for some dataflow problem D, which induces a partial order on dataflow solutions to the entire graph. By examining this semi-lattice, we can understand the relationship between the different potential solutions to a dataflow problem.

Suppose we consider the function $F : S \to S$, where $S = D_d^n$, where D_d is the domain of the dataflow problem D, n is the number of basic blocks in the program, and S is the set of all tuples of possible values for values of basic block entry points. In other words, S is the set of all possible values that basic block entry points of D could take on at any time.

F itself will map from one set of basic block entry points to another, based on the meet operator and transfer function of D (at a high level, imagine F applies a single iteration of the iterative dataflow algorithm specified by D). Assume a forward algorithm.

From the way we defined F, we know that any fixed point, d to F is a solution to our dataflow problem D, since we can extract from d the values of basic block entry points, apply our transfer function to each basic block, obtain values of basic block exit points, and since $F(d) = d$, the values of basic blocks in the iterative algorithm will not change. (Another way to think about this is that if F maps to one iteration of the iterative algorithm, so if $F(d) = d$, this implies d is a solution by definition).

We know that a dataflow problem can have multiple fixed point solutions. Suppose M is the solution obtained from our dataflow algorithm (applying F repeatedly), and d is any other fixed point to D. Show that $M \geq d$. You may ignore the initialization of boundary points. *Hint: Recall that the we initialize interior points to $\top$ and that our transfer function is monotonic.*

Since we initialize the interior points to $\top$ in our dataflow algorithm, the sequence of basic block entry points we will see are $\top, F(\top), F^2(\top), ...$ till convergence. On convergence, we know the $M = F^m(\top)$, for some finite m. This is because we assume our domain only has finite descending chains and that F is defined with a monotonic transfer function, so applying F to $\top$ cannot descend indefinitely.

Now since we initialized our interior points to $I = \top$, we know that $\top \geq d$ for any fixed point solution d. Since our transfer function is monotonic, $F(\top) \geq F(d)$; however, note that $F(d) = d$, which implies that $F(\top) \geq F(d) = d$ so $F(I) \geq d$. Using induction, we assume that $F^{k-1}(I) \geq d$, then since F is monotonic, we know that $F(F^{k-1}(I)) \geq F(d)$. However, $F(F^{k-1}(I)) = F^k(I)$ and $F(d) = d$ since d is a fixed point of F. Therefore, $F^k(I) \geq d$ for any k.

Therefore, we can conclude that $F^m(I) = F^m(\top) \geq d$ which proves $M \geq d$.

Note that the initialization to $\top$ was crucial for our induction, since if we initialize to $I = d < \top$, then convergence to the MFP is no longer guaranteed.