Program Semantics

Xiangyu Zhang

Language

Program P ::= s

Statement s ::= s1; s2 | x= y | x = y op z | x= c |
if (x) s1 else s2 |
while (x) s

Operation op :: = + | - | * | / | > | < | …

Value c :: = 0 | 1 | 2 … | true | false

Variable x, x1, x2, x3

Configuration

- $< s, \delta > \rightarrow < s', \delta >$
  - Store δ : Variable -> Value

Semantics Rules

$$\frac{\delta' = \delta[x \mapsto c]}{< x = c; s, \delta > \rightarrow < s, \delta' >}$$

Const-Assign

$$\frac{\delta' = \delta[x \mapsto \delta[y]]}{< x = y; s, \delta > \rightarrow < s, \delta' >}$$

Copy

$$\frac{a = \delta[y] \quad b = \delta[z] \quad c = a + b \qquad \delta' = \delta[x \mapsto c]}{< x = y + z; s, \delta > \rightarrow < s, \delta' >}$$

BinOp-Add

Semantics Rules

$$\frac{\delta[x] = true}{< if\ (x)\ s1\ else\ s2; s, \delta > \rightarrow < s1; s, \delta >}$$ If-T

$$\frac{\delta[x] = false}{< if\ (x)\ s1\ else\ s2; s, \delta > \rightarrow < s2; s, \delta >}$$ If-F

$$\frac{}{< while\ (x) s1; s, \delta > \rightarrow < if\ (x)\ s1; while\ (x)\ s1\ else\ skip; s, \delta >}$$ While

Example

```
i=0;
sum=0;
N=2;
while (i<N) {
    sum=sum+i;
    i=i+1;
}
```

Extend the Language with Pointers

Program P ::= s

Statement s ::= s1; s2 | x= y | x = y op z | x= c |
x= &y | (*x)=y | x= (*y)
if (x) s1 else s2 |
while (x) s

Operation op :: = + | - | * | / | > | < | …

Value c :: = 0 | 1 | 2 … | true | false

Address a :: = 0 | 1 | 2…

Variable x, x1, x2, x3

Configuration

- $< s, \delta > \rightarrow < s', \delta >$
  - Store δ : Address -> Value
  - Assume there is a predefined mapping α: Variable -> Address

Semantics Rules

$$\frac{\delta' = \delta[\alpha[x] \mapsto c]}{< x = c; s, \delta > \rightarrow < s, \delta' >}$$

Const-Assign

$$\frac{\delta' = \delta[\alpha[x] \mapsto \delta[\alpha[y]]]}{< x = y; s, \delta > \rightarrow < s, \delta' >}$$

Copy

$$\frac{a = \delta[\alpha[y]] \quad b = \delta[\alpha[z]] \quad c = a + b \qquad \delta' = \delta[\alpha[x] \mapsto c]}{< x = y + z; s, \delta > \rightarrow < s, \delta' >}$$

BinOp-Add

Semantics Rules

$$\frac{\delta' = \delta[\alpha[x] \mapsto a] \quad a = \alpha[y]}{< x = \&y; s, \delta > \rightarrow < s, \delta' >}$$

Addr-Of

$$\frac{\delta' = \delta[\delta[\alpha[x]] \mapsto \delta[\alpha[y]]]}{< (*x) = y; s, \delta > \rightarrow < s, \delta' >}$$

Ptr-Write

$$\frac{\delta' = \delta[\alpha[x] \mapsto \delta[\delta[\alpha[y]]]]}{< x = *y; s, \delta > \rightarrow < s, \delta' >}$$

Ptr-Read

Extend the Language with Heap

Program P ::= s

Statement s ::= s1; s2 | x= y | x = y op z | x= c |
x= &y | (*x)=y | x=(*y) | x=malloc (y)
if (x) s1 else s2 |
while (x) s

Operation op :: = + | - | * | / | > | < | …

Value c :: = 0 | 1 | 2 … | true | false

Address a :: = 0 | 1 | 2…

Variable x, x1, x2, x3

Configuration

- $< s, \delta, \gamma > \rightarrow < s', \delta, \gamma' >$
  - Store δ : Address -> Value
  - HeapTop γ: Int
    - Initinally, $\gamma = |\alpha|$
  - Assume there is a predefined mapping α: Variable -> Address

Semantics Rules

$$\frac{\delta' = \delta[\alpha[x] \mapsto c]}{< x = c; s, \delta, \gamma > \rightarrow < s, \delta', \gamma >}$$

Const-Assign

$$\frac{\delta' = \delta[\alpha[x] \mapsto \delta[\alpha[y]]]}{< x = y; s, \delta, \gamma > \rightarrow < s, \delta', \gamma >}$$

Copy

$$\frac{a = \delta[\alpha[y]] \quad b = \delta[\alpha[z]] \quad c = a + b \qquad \delta' = \delta[\alpha[x] \mapsto c]}{< x = y + z; s, \delta, \gamma > \rightarrow < s, \delta', \gamma >}$$

BinOp-Add

Semantics Rules

$$\frac{\delta' = \delta[\alpha[x] \mapsto a] \quad a = \alpha[y]}{< x = \&y; s, \delta, \gamma > \rightarrow < s, \delta', \gamma >}$$

Addr-Of

$$\frac{\delta' = \delta[\delta[\alpha[x]] \mapsto \delta[\alpha[y]]]}{< (*x) = y; s, \delta, \gamma > \rightarrow < s, \delta', \gamma >}$$

Ptr-Write

$$\frac{\delta' = \delta[\alpha[x] \mapsto \delta[\delta[\alpha[y]]]]}{< x = *y; s, \delta, \gamma > \rightarrow < s, \delta', \gamma >}$$

Ptr-Read

$$\frac{\delta' = \delta[\alpha[x] \mapsto \gamma] \quad \gamma' = \gamma + \delta[\alpha[y]]}{< x = malloc(y); s, \delta, \gamma > \rightarrow < s, \delta', \gamma' >}$$

Malloc

Extend the Language with Functions

Program P ::= fd; s
Function f :: = M(y) { s }
FuncDef fd :: = f; f
FuncId M, M1, M2, …
Statement s ::= s1; s2 | x= y | x = y op z | x= c |
if (x) s1 else s2 |
while (x) s | call(M, x)
Operation op :: = + | - | * | / | > | < | …
Value c :: = 0 | 1 | 2 … | true | false
Variable x, x1, x2, x3

Formalizing Dynamic Analysis

A Dynamic Checker for Heap Overflow

- Check if a heap access dereferences an address beyond the allocated buffer

```
p=malloc (10);
x=p+2;
q=x+9;
(*q)=…
```

Heap Language

Program P ::= s

Statement s ::= s1; s2 | x= y | x = y op z | x= c |
x= &y | (*x)=y | x=*y | x=malloc (y)
if (x) s1 else s2 |
while (x) s

Operation op :: = + | - | * | / | > | < | …

Value c :: = 0 | 1 | 2 … | true | false

Address a :: = 0 | 1 | 2…

Variable x, x1, x2, x3

A Plausible Solution - Configuration

- $< s, \delta, \gamma, H > \rightarrow < s', \delta, \gamma', H' >$
  - Store δ : Address -> Value
  - Heap H: Variable -> Address × Int
  - HeapTop γ: Int
    - Initinally, $\gamma = |\alpha|$
  - Assume there is a predefined mapping α: Variable -> Address

Semantics Rules

$$\frac{\delta' = \delta[\alpha[x] \mapsto \gamma] \quad \gamma' = \gamma + \delta[\alpha[y]] \quad H' = H[x \mapsto < \gamma, \delta[\alpha[y]] >]}{< x = malloc(y); s, \delta, \gamma, H > \rightarrow < s, \delta', \gamma', H' >}$$

Malloc

$$\frac{\delta' = \delta\left[\delta[\alpha[x]] \mapsto \delta[\alpha[y]]\right] \quad H[x] =< a, i > \quad a \leq \delta[\alpha[x]] < a + i}{< (*x) = y; s, \delta, \gamma, H > \rightarrow < s, \delta', \gamma, H >}$$

Ptr-Write

Semantics Rules

$$\frac{\delta' = \delta\left[\delta[\alpha[x]] \mapsto \delta[\alpha[y]]\right] \quad H[x] =< a, i > \qquad a + i \leq \delta[\alpha[x]] \quad \vee \quad \delta[\alpha[x]] < a}{< (*x) = y; s, \delta, \gamma, H > \rightarrow < exception, \delta', \gamma, H >}$$

Ptr-Write-Excp

-Is it correct? If not, how to improve?

Another Solution - Configuration

- $< s, \delta, \gamma, H > \rightarrow < s', \delta, \gamma', H' >$
  - Store δ : Address -> Value
  - Heap H: Address -> Address × Int
  - HeapTop γ: Int
    - Initinally, $\gamma = |\alpha|$
  - Assume there is a predefined mapping α: Variable -> Address

Semantics Rules

$$\frac{\delta' = \delta[\alpha[x] \mapsto c]}{< x = c; s, \delta, \gamma, H > \rightarrow < s, \delta', \gamma, H >}$$ Const-Assign

$$\frac{\delta' = \delta\left[\alpha[x] \mapsto \delta[\alpha[y]]\right] \; H' = H[\alpha[x] \mapsto H[\alpha[y]]]}{< x = y; s, \delta, \gamma, H > \rightarrow < s, \delta', \gamma, H' >}$$ Copy

$$\frac{\begin{array}{c} a = \delta[\alpha[y]] \quad b = \delta[\alpha[z]] \quad c = a + b \\ H[\alpha[x]] = < a, i > \quad \delta' = \delta[\alpha[x] \mapsto c] \\ H' = H[\alpha[x] \mapsto < a, i >] \end{array}}{< x = y + z; s, \delta, \gamma, H > \rightarrow < s, \delta', \gamma, H' >}$$ Pnt-Add-z

Semantics Rules

$$\frac{a = \delta[\alpha[y]] \quad b = \delta[\alpha[z]] \quad c = a + b \qquad H[\alpha[y]] = < a, i > \quad \delta' = \delta[\alpha[x] \mapsto c] \qquad H' = H[\alpha[x] \mapsto < a, i >]}{< x = y + z; s, \delta, \gamma, H > \rightarrow < s, \delta', \gamma, H' >}$$

Pnt-Add-y

$$\frac{a = \delta[\alpha[y]] \quad b = \delta[\alpha[z]] \quad c = a + b \qquad H[\alpha[y]] = undef \quad \delta' = \delta[\alpha[x] \mapsto c] \qquad H[\alpha[z]] = undef}{< x = y + z; s, \delta, \gamma, H > \rightarrow < s, \delta', \gamma, H >}$$

NonPnt-Add

Semantics Rules

$$\frac{\delta' = \delta[\alpha[x] \mapsto \gamma] \quad \gamma' = \gamma + \delta[\alpha[y]] \quad H' = H[\delta[\alpha[x]] \mapsto < \gamma, \delta[\alpha[y]] >]}{< x = malloc(y); s, \delta, \gamma, H > \rightarrow < s, \delta', \gamma', H' >}$$

Malloc

$$\frac{\delta' = \delta\left[\delta[\alpha[x]] \mapsto \delta[\alpha[y]]\right] \; H\left[\delta[\alpha[x]]\right] = < a, i > \quad a \leq \delta[\alpha[x]] < a + i \quad H' = H[\delta[\alpha[x]] \mapsto H[\delta[\alpha[y]]]]}{< (* x) = y; s, \delta, \gamma, H > \rightarrow < s, \delta', \gamma, H' >}$$

Ptr-Write

Example Revisit

```
p=malloc (10);
x=p+2;
q=x+9;
(*q)=…
```

Soundness Proof

- For each memory access, if the address exceeds the bound of the corresponding buffer, the execution must terminate with an exception.
  - Prove by induction

Enhance the Analysis to Detect Dangling Pointers

Dynamic Data Dependence Detection

- We aim to detect data dependence on the fly

Dynamic Data Dependence Detection

- We aim to detect data dependence on the fly

Heap Language

Program P ::= s

Statement s ::= s1; s2 | $x =^{L} y$ | $x =^{L} y$ op z | $x =^{L} c$ |
$x =^{L} \&y$ | $(*x) =^{L} y$ | $x =^{L} *y$ | $x =^{L}$ malloc (y)
if (x^{L}) s1 else s2 |
while (x^{L}) s

Operation op :: = + | - | * | / | > | < | …

Value c :: = 0 | 1 | 2 … | true | false

Address a :: = 0 | 1 | 2…

Variable x, x1, x2, x3

Label L, L1, L2,…

Configuration

- $< s, \delta, \gamma, C, D, X > \rightarrow < s', \delta', \gamma', C', D', X' >$
  - Counter C: Label -> Int
  - Definition D: Address -> Label × Int
  - Dependences X: P (Label × Int × Label × Int)

Semantics Rules

$$\frac{\delta' = \delta\left[\alpha[x] \mapsto \delta[\alpha[y]]\right] \quad C' = C[L \mapsto C[L] + 1] \quad D' = D[\alpha[x] \mapsto < L, C[L] >] \quad X' = X \cup < L, C[L], D[\alpha[y]] >}{< x = {}^{L}y; s, \delta, \gamma, C, D, X > \rightarrow < s, \delta', \gamma, C', D', X' >}$$

Copy

$$\frac{\delta' = \delta\left[\delta[\alpha[x]] \mapsto \delta[\alpha[y]]\right] \quad C' = C[L \mapsto C[L] + 1] \quad D' = D[\delta[\alpha[x]] \mapsto < L, C[L] >] \quad X' = X \cup < L, C[L], D[\alpha[y]] > \cup < L, C[L], D[\alpha[x]] >}{< (* x) = {}^{L}y; s, \delta, \gamma, C, D, X > \rightarrow < s, \delta', \gamma, C', D'X' >}$$

Ptr-Write

Dynamic Control Dependence Detection

- We aim to detect dynamic control dependence on the fly

Dual Execution Semantics

- How to align two executions with slightly different inputs?
  - Critical for debugging