

Analysis Correctness

Reading: NNH 2.2 (optional)

17-654/17-765

Analysis of Software Artifacts

Jonathan Aldrich

Announcements

- Office Hours
 - Nicholas Sherman
 - **This week: Wednesday 3pm, MSE Cave**
 - Future: Tuesday 4pm, MSE Cave
 - Dean Sutherland
 - Thursday 4pm, Wean Hall 8130
 - Jonathan Aldrich
 - Wednesday 1pm, Wean Hall 8212

What does Correctness Mean?

What does Correctness Mean?

- Intuition
 - At a fixed point, analysis results are a *conservative abstraction* of *program execution*
 - *program execution* must be formally defined
 - *abstraction* relates program execution to data flow values
 - *conservative* means $\text{truth} \sqsubseteq \text{analysis results}$

Execution Traces

- Sequence of $\langle \text{pp}, \text{mem} \rangle$ pairs
 - pp is a program point
 - Just before statement pp
 - mem is the state of variables in memory

```

[y := x]1;
[z := 1]2;
while [y > 1]3 do
  [z := z * y]4;
  [y := y - 1]5;
[y := 0]6;

```

<u>pp</u>	<u>x</u>	<u>y</u>	<u>z</u>
1	2	0	0
2	2	2	0
3	2	2	1
4	2	2	1
5	2	2	2
3	2	1	2
6	2	1	2
-	2	0	2

Execution Traces

- Sequence of $\langle pp, mem \rangle$ pairs
 - pp is a program point
 - Just before statement pp
 - mem is the state of variables in memory

<u>pp</u>	<u>x</u>	<u>y</u>	<u>z</u>
1	1	0	0
2	1	1	0
3	1	1	1
6	1	1	1
-	1	0	1

```
[y := x]1;  
[z := 1]2;  
while [y > 1]3 do  
  [z := z * y]4;  
  [y := y - 1]5;  
[y := 0]6;
```

Execution Traces

- Sequence of $\langle \text{pp}, \text{mem} \rangle$ pairs
 - pp is a program point
 - Just before statement pp
 - mem is the state of variables in memory

$[y := x]^1;$

$[z := 1]^2;$

while $[y > 1]^3$ do

$[z := z * y]^4;$

$[y := y - 1]^5;$

$[y := 0]^6;$

<u>pp</u>	<u>x</u>	<u>y</u>	<u>z</u>
1	3	0	0
2	3	3	0
3	3	3	1
4	3	3	1
5	3	3	3
3	3	2	3
4	3	2	3
5	3	2	6
3	3	1	6
6	3	1	6
-	3	0	6

Execution Traces

- Sequence of $\langle pp, mem \rangle$ pairs
 - pp is a program point
 - Just before statement pp
 - mem is the state of variables in memory

pp x y z

Repeat for all possible initial values of $x, y, z!$

```
[y := x]1;  
[z := 1]2;  
while [y > 1]3 do  
  [z := z * y]4;  
  [y := y - 1]5;  
[y := 0]6;
```


Abstraction

- Abstraction function α
 - maps traces to data flow values at a certain time t in the trace

- $\alpha_{CP}(\langle p_1, m_1 \rangle \dots \langle p_n, m_n \rangle, t)$
 $= m_t$

- Also define program point function pp

- $pp(\langle p_1, m_1 \rangle \dots \langle p_n, m_n \rangle, t)$
 $= p_t$

<u>t</u>	<u>pp</u>	<u>x</u>	<u>y</u>	<u>z</u>
0	1	3	0	0
1	2	3	3	0
2	3	3	3	1
3	4	3	3	1
4	5	3	3	3
5	3	3	2	3
6	4	3	2	3
7	5	3	2	6
8	3	3	1	6
9	6	3	1	6
10	-	3	0	6

$$\alpha_{CP}(T, 0) = (x=3, y=0, z=0)$$

$$\alpha_{CP}(T, 10) = (x=3, y=0, z=6)$$

What does Correctness Mean?

- Intuition
 - At a fixed point, analysis results are a *conservative abstraction of program execution*
- *Soundness* condition
 - When data flow analysis reaches a fixed point F , then for all traces T and all times t in each trace, $\alpha(T, t) \sqsubseteq F(pp(T, t))$
 - Constant propagation
 - For trace on last slide with $t=10$
 - $\alpha_{CP}(T, 10) = \langle x=3, y=0, z=6 \rangle$
 - $F_{CP}(pp(T, 10)) = F_{CP}(\text{exit}_6) = \langle x=\top, y=0, z=\top \rangle$
 - $\langle x=3, y=0, z=6 \rangle \sqsubseteq_{\top} \langle x=\top, y=0, z=\top \rangle$
 - Because $3 \sqsubseteq \top$ and $0 \sqsubseteq 0$ and $6 \sqsubseteq \top$ in the CP lattice
 - To prove soundness, repeat for all times in all traces

Why care about Soundness?

- Analysis Producers
 - Writing analyses is hard
 - People make mistakes all the time
 - Need to know how to *think* about correctness
 - When the analysis gets tricky, it's useful to prove it correct formally
- Analysis Consumers
 - Sound analysis provides guarantees
 - Optimizations won't break the program
 - Finds all defects of a certain sort
 - Decision making
 - Knowledge of soundness techniques lets you ask the right questions about a tool you are considering
 - Soundness affects where you invest QA resources
 - Focus testing efforts on areas where you don't have a sound analysis!

Proving Soundness

- Formally define analysis
 - We already know how
- Formalize trace semantics
- Define abstraction function
- Prove *local soundness* for flow functions
- Apply *global soundness theorem*

Semantics of WHILE Expressions

store σ has type **State** = map from **Var** to \mathbb{Z}

- \mathbb{Z} the set of integers; we assume no boolean-typed vars

$\mathcal{A}(\mathbf{AExp}, \mathbf{State})$ computes the value of **AExp** in **State**

$$\mathcal{A}(x, \sigma) = \sigma(x)$$

$$\mathcal{A}(n, \sigma) = n$$

$$\mathcal{A}(a_1 \text{ op}_a a_2, \sigma) = \mathcal{A}(a_1, \sigma) \text{ op}_a \mathcal{A}(a_2, \sigma)$$

Example (assume $\sigma = (x=5, y=7)$)

$$\begin{aligned} \mathcal{A}(x+3, \sigma) &= \mathcal{A}(x, \sigma) + \mathcal{A}(3, \sigma) \\ &= \sigma(x) + 3 \\ &= 5 + 3 \\ &= 8 \end{aligned}$$

Semantics of WHILE Expressions

store σ has type **State** = map from **Var** to \mathbb{Z}

- \mathbb{Z} the set of integers; we assume no boolean-typed vars

$\mathcal{A}(\mathbf{AExp}, \mathbf{State})$ computes the value of **AExp** in **State**

$$\mathcal{A}(x, \sigma) = \sigma(x)$$

$$\mathcal{A}(n, \sigma) = n$$

$$\mathcal{A}(a_1 \text{ op}_a a_2, \sigma) = \mathcal{A}(a_1, \sigma) \text{ op}_a \mathcal{A}(a_2, \sigma)$$

$\mathcal{B}(\mathbf{BExp}, \mathbf{State})$ computes whether **BExp** is true in **State**

$$\mathcal{B}(\text{not } b, \sigma) = \text{not } \mathcal{B}(b, \sigma)$$

$$\mathcal{B}(b_1 \text{ op}_b b_2, \sigma) = \mathcal{B}(b_1, \sigma) \text{ op}_b \mathcal{B}(b_2, \sigma)$$

$$\mathcal{B}(a_1 \text{ op}_r a_2, \sigma) = \mathcal{A}(a_1, \sigma) \text{ op}_r \mathcal{A}(a_2, \sigma)$$

Semantics of Assignment in WHILE

$$\frac{}{([x := a]^\ell, \sigma) \rightarrow ([], \sigma[x \mapsto \mathcal{A}(a, \sigma)])} [ass]$$

- Start with a program $[x := a]^\ell$ and a store σ
 - Goal: rewrite to a new program and new store
- We execute $[x := a]^\ell$ resulting in:
 - The empty program $[]$
 - Evaluate a with store σ to get $\mathcal{A}(a, \sigma)$
 - Update x 's value to be $\mathcal{A}(a, \sigma)$
- Example: $a = x+3$, $\sigma = (x=5, y=7)$
 - We get the pair $([], (x=8, y=7))$

Semantics of WHILE Statements

$$\frac{}{([x := a]^\ell, \sigma) \rightarrow ([], \sigma[x \mapsto \mathcal{A}(a, \sigma)])} [ass]$$

$$\frac{}{([skip]^\ell, \sigma) \rightarrow ([], \sigma)} [skip]$$

$$\frac{(S_1, \sigma) \rightarrow (S'_1, \sigma') \quad S'_1 \neq []}{(S_1; S_2, \sigma) \rightarrow (S'_1; S_2, \sigma')} [seq1]$$

$$\frac{(S_1, \sigma) \rightarrow ([], \sigma')}{(S_1; S_2, \sigma) \rightarrow (S_2, \sigma')} [seq2]$$

$$\frac{\mathcal{B}(b, \sigma) = \text{true}}{(\text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2, \sigma) \rightarrow (S_1, \sigma)} [if1]$$

$$\frac{\mathcal{B}(b, \sigma) = \text{false}}{(\text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2, \sigma) \rightarrow (S_2, \sigma)} [if2]$$

$$\frac{\mathcal{B}(b, \sigma) = \text{true}}{(\text{while } [b]^\ell \text{ do } S, \sigma) \rightarrow (S; \text{while } [b]^\ell \text{ do } S, \sigma)} [while1]$$

$$\frac{\mathcal{B}(b, \sigma) = \text{false}}{(\text{while } [b]^\ell \text{ do } S, \sigma) \rightarrow ([], \sigma)} [while2]$$

Execution in WHILE

$$\frac{}{([x := a]^{\ell}, \sigma) \rightarrow ([], \sigma[x \mapsto \mathcal{A}(a, \sigma)])} [ass]$$
$$\frac{(S_1, \sigma) \rightarrow ([], \sigma')}{(S_1; S_2, \sigma) \rightarrow (S_2, \sigma')} [seq_2]$$

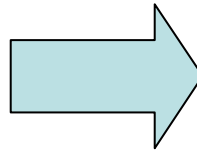
([y := x]¹; [z := 1]²;

while [y>1]³ do

[z := z * y]⁴; [y := y - 1]⁵;

[y := 0]⁶;

(x=3,y=0,z=0) **)**



([z := 1]²;

while [y>1]³ do

[z := z * y]⁴; [y := y - 1]⁵;

[y := 0]⁶;

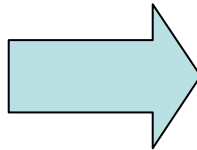
(x=3,y=3,z=0) **)**

Execution in WHILE

$$\frac{}{([x := a]^{\ell}, \sigma) \rightarrow ([], \sigma[x \mapsto \mathcal{A}(a, \sigma)])} [ass]$$

$$\frac{(S_1, \sigma) \rightarrow ([], \sigma')}{(S_1; S_2, \sigma) \rightarrow (S_2, \sigma')} [seq_2]$$

([z := 1]²;
 while [y>1]³ do
 [z := z * y]⁴; [y := y - 1]⁵;
 [y := 0]⁶;
 (x=3,y=3,z=0))



(while [y>1]³ do
 [z := z * y]⁴; [y := y - 1]⁵;
 [y := 0]⁶;
 (x=3,y=3,z=1))

Execution in WHILE

$$\frac{\mathcal{B}(b, \sigma) = \text{true}}{(\text{while } [b]^\ell \text{ do } S, \sigma) \rightarrow (S; \text{while } [b]^\ell \text{ do } S, \sigma)} \text{ [while}_1\text{]}$$

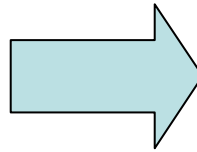
$$\frac{(S_1, \sigma) \rightarrow (S'_1, \sigma') \quad S'_1 \neq []}{(S_1; S_2, \sigma) \rightarrow (S'_1; S_2, \sigma')} \text{ [seq}_1\text{]}$$

(while [y>1]³ do

[z := z * y]⁴; [y := y - 1]⁵;

[y := 0]⁶;;

(x=3,y=3,z=1))



([z := z * y]⁴; [y := y - 1]⁵;

while [y>1]³ do

[z := z * y]⁴; [y := y - 1]⁵;

[y := 0]⁶;;

(x=3,y=3,z=1))

Execution in WHILE

$$\frac{}{([x := a]^{\ell}, \sigma) \rightarrow ([], \sigma[x \mapsto \mathcal{A}(a, \sigma)])} [ass]$$

$$\frac{(S_1, \sigma) \rightarrow ([], \sigma')}{(S_1; S_2, \sigma) \rightarrow (S_2, \sigma')} [seq_2]$$

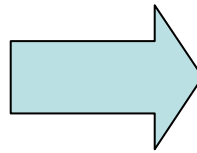
([z := z * y]⁴; [y := y - 1]⁵;

while [y > 1]³ do

 [z := z * y]⁴; [y := y - 1]⁵;

[y := 0]⁶;;

(x=3,y=3,z=1) **)**



([y := y - 1]⁵;

while [y > 1]³ do

 [z := z * y]⁴; [y := y - 1]⁵;

[y := 0]⁶;;

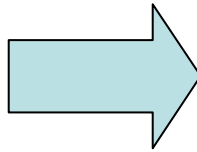
(x=3,y=3,z=3) **)**

Execution in WHILE

$$\frac{}{([x := a]^{\ell}, \sigma) \rightarrow ([], \sigma[x \mapsto \mathcal{A}(a, \sigma)])} [ass]$$

$$\frac{(S_1, \sigma) \rightarrow ([], \sigma')}{(S_1; S_2, \sigma) \rightarrow (S_2, \sigma')} [seq_2]$$

([y := y - 1]⁵;
 while [y > 1]³ do
 [z := z * y]⁴; [y := y - 1]⁵;
 [y := 0]⁶;
 (x=3, y=3, z=3))



(while [y > 1]³ do
 [z := z * y]⁴; [y := y - 1]⁵;
 [y := 0]⁶;
 (x=3, y=2, z=3))

WHILE Traces, Formally

- A trace for program S_1 and initial state σ_1 is either:
 - a finite sequence $(S_1, \sigma_1), \dots, ([], \sigma_n)$,
where $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1})$ for $i \in 1, \dots, n-1$
 - an infinite sequence $(S_1, \sigma_1), \dots, (S_i, \sigma_i), \dots$
where $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1})$ for $i \geq 1$
- Slight notational simplification
 - We will abbreviate $(S_1, \sigma_1), \dots, (S_n, \sigma_n)$
as $(\text{first}(S_1), \sigma_1), \dots, (\text{first}(S_n), \sigma_n)$
 - Uses program counter labels instead of complete programs