

## Context Sensitivity

Lecture 20

Prof. Aiken & Dill CS 357 Lecture 20

1

## Monomorphic Analysis

$$f(x) = x$$

$$g() = f(1)$$

$$h() = f(2)$$

What will a monomorphic analysis report about the possible return values of  $g$  and  $h$ ?

Prof. Aiken & Dill CS 357 Lecture 20

2

## What's Going On?

- A monomorphic analysis
  - Summarizes over all call sites
  - Produces one abstract value for the result of the function
  - Which is the value of every call to the function

Prof. Aiken & Dill CS 357 Lecture 20

3

## Context Sensitivity

- In many cases we want analysis to be *context sensitive*
- In the example, we want the analysis to be specialized to what we know at the call sites in  $g$  and  $h$  separately

Prof. Aiken & Dill CS 357 Lecture 20

4

## Idea #1: Inlining/Cloning

$$f(x) = x$$

$$g() = f(1)$$

$$h() = f(2)$$

$$f_1(x) = x$$

$$f_2(x) = x$$

$$g() = f_1(1)$$

$$h() = f_2(2)$$

Prof. Aiken & Dill CS 357 Lecture 20

5

## An Aside About Type Systems

- Polymorphic type systems essentially do inlining/cloning
  - But at the level of types

$$f(x) = x$$

$$g() = f(1)$$

$$h() = f(2)$$

$$\text{forall } a. a \rightarrow a$$

$$(a \rightarrow a)[1/a]$$

$$(a \rightarrow a)[2/a]$$

Prof. Aiken & Dill CS 357 Lecture 20

6

**Infeasible Paths**

- Some of the flow paths in the monomorphic analysis are *infeasible*
  - Represent combinations of calls/returns that can never happen in a real execution

$f(x) = x$

$g() = f(1)$   
 $h() = f(2)$

**Infeasible Paths (Cont)**

- We want to rule out infeasible paths
- First make explicit which paths are feasible
  - Assume we have a distinct name for each call site  $c$
  - Label  $c$ 's call edge with  $(c$
  - Label  $c$ 's return edge with  $)c$

$f(x) = x$   
 $g() = f(1)$   
 $h() = f(2)$

**Feasible Paths**

- The *feasible paths* are those where the calls and returns match and are properly nested
  - Information is only returned to a call site if the flow path passed through the call to that site

$f(x) = x$   
 $g() = f(1)$   
 $h() = f(2)$

**Recursive Functions**

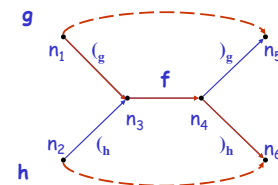
- This even works with recursive functions

$f(x) = \dots f(y) \dots f(z) \dots x$   
 $g() = f(1)$   
 $h() = f(2)$

**Summary**

- We want to analyze only the feasible paths
- These are the paths in which calls & returns are properly matched

**Graph Reachability**



Observation:  
 A closed context-sensitive graph has fewer edges than a closed monomorphic graph.

### Idea

Use constructors to record the stack of function calls.

(<sub>i</sub> edges:  $o_i(X) \subseteq Y$ )  
 )<sub>i</sub> edges:  $X \subseteq o_i(Y)$

Prof. Aiken & Dill CS 357 Lecture 20 13

### Idea

Use constructors to record the stack of function calls.

(<sub>i</sub> edges:  $o_i(X) \subseteq Y$ )  
 )<sub>i</sub> edges:  $X \subseteq o_i(Y)$

Prof. Aiken & Dill CS 357 Lecture 20 14

### Problem: This Works Too Well . . .

Consider the constraints  
 $o_g(n_1) \subseteq n_3 \subseteq n_4 \subseteq o_h(n_1)$

These constraints have no solution.

Prof. Aiken & Dill CS 357 Lecture 20 15

### A Problem: This Works Too Well . . .

- Set constraints can be inconsistent

Constraints:

$$o_1(X) \subseteq Y$$

$$Y \subseteq o_2(Z)$$

CFL Graph:

↑  
 These constraints have no solution!

Prof. Aiken & Dill CS 357 Lecture 20 16

### A Brief Digression

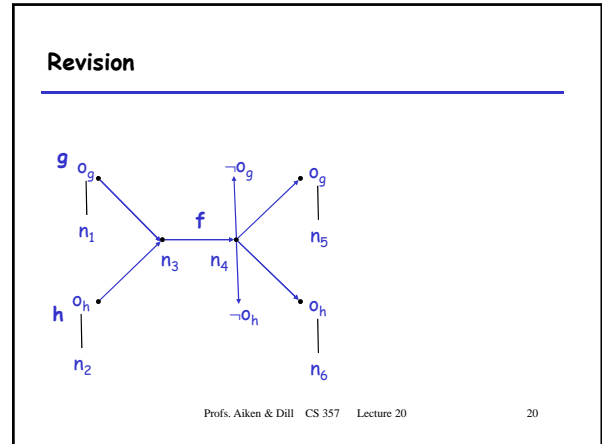
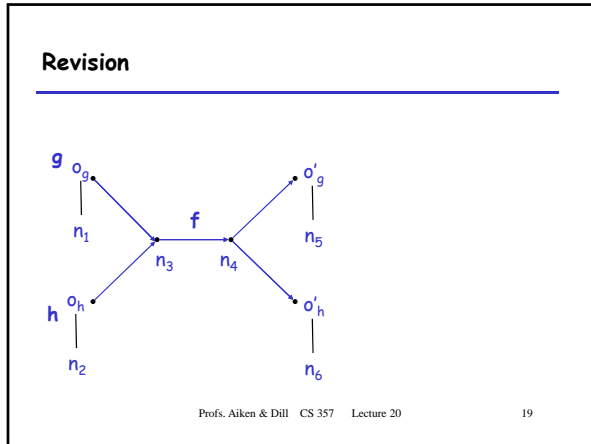
- A union is *discriminative* if each disjunct has a distinct head constructor
  - $c(X,Y) \cup d(W,U)$  is discriminative
  - $c(X,Y) \cup c(W,U)$  is not discriminative
- Constraints w/discriminative unions are easy:
 
$$c(A,B) \subseteq c(X,Y) \cup d(W,U) \Leftrightarrow c(A,B) \subseteq c(X,Y)$$

Prof. Aiken & Dill CS 357 Lecture 20 17

### Definition

$$o'(x) = o(x) \cup \neg o(1)$$

Prof. Aiken & Dill CS 357 Lecture 20 18



### The Reduction in Detail

- Encoding initial edges:
  - $x \xrightarrow{\quad} y \quad X \subseteq Y$
  - $x \xrightarrow{(}_i \quad y \quad o_i(X) \subseteq Y$
  - $x \xrightarrow{)}_i \quad y \quad X \subseteq o'_i(Y)$
- A technical requirement on each variable:  $c_x \subseteq X$
- Theorem: there is a matched parenthesis path from  $x$  to  $y$  in the graph iff  $c_x \subseteq Y$

Prof. Aiken & Dill CS 357 Lecture 20 21

### (Dyck) CFL Reachability

Graph:

Grammar:

$$S ::= \epsilon$$

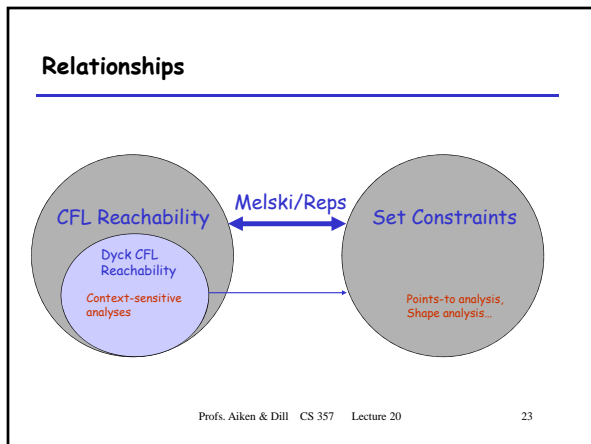
$$| s$$

$$| (i S)_i$$

$$| S S$$

Goal: Find path labeled by words in the grammar  
 Intuition: Matching requirement filters infeasible paths

Prof. Aiken & Dill CS 357 Lecture 20 22



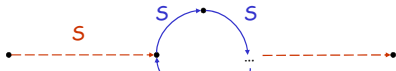
### Optimizations and Extensions

- Cycle Elimination
- PN Reachability
- See paper for more...
  - Inductive form
  - Global nodes
  - Clustering

Prof. Aiken & Dill CS 357 Lecture 20 24

### Cycle Elimination

- Variables in a cycle are all equivalent  
 $X_1 \subseteq X_2 \dots \subseteq X_n \subseteq X_1$
- Optimization: collapse them into one variable
- True for a cycle of S-edges in a Dyck graph
  - Can always concatenate S onto any word



Prof. Aiken & Dill CS 357 Lecture 20 25

### PN Reachability

- Some apps need substrings of matched strings  
 $((()))()()()((()))$
- Consider finding prefixes...
  - A matched path with a  $((_k)_k$  tail
  - There is a prefix path from  $x$  to  $y$  iff a term  $o_k(o_j(o_i(c_x)))$  in  $Y$
  - A bit more tedious than handling fully matched paths

Prof. Aiken & Dill CS 357 Lecture 20 26

### Case Study: Tainting Analysis

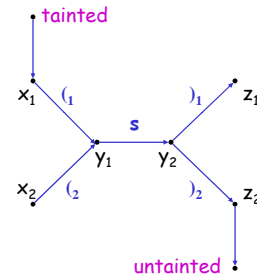
- Can values under adversarial control reach trusted data?
- Use type qualifiers
  - tainted**: Data came from an untrusted source
  - untainted**: Data must be trusted
- Polymorphic type qualifier inference

Prof. Aiken & Dill CS 357 Lecture 20 27

### Tainting Analysis: Example

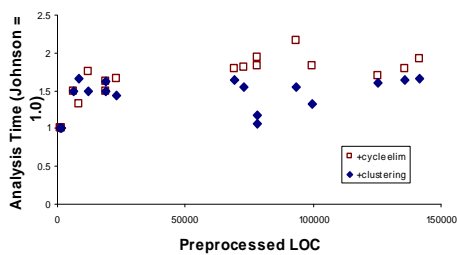
```
int id(int y1)
{
  int y2 = y1;
  return y2;
}

int main()
{
  tainted int x1;
  int z1, x2;
  tainted int z2;
  z1 = id(x1); // call site 1
  z2 = id(x2); // call site 2
}
```



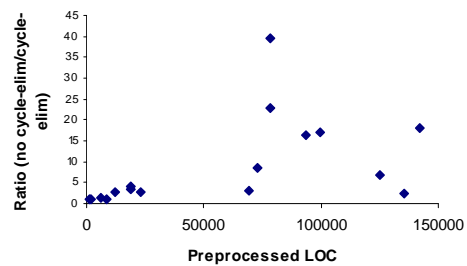
Prof. Aiken & Dill CS 357 Lecture 20 28

### Results



Prof. Aiken & Dill CS 357 Lecture 20 29

### Effects of Cycle Elimination



Prof. Aiken & Dill CS 357 Lecture 20 30

**Once More, From the Top...**

- Why not use the Melski-Reps reduction?
- Requires local encoding of grammar
  - One constraint per node per production
- Doesn't expose cycles
  - Or other optimization opportunities

**Another Idea**

- Why not use set constraints to
  - describe recursive data structures
    - Lists, trees, etc.
  - describe recursive control
    - context-free reachability
- Get fully context-sensitive, data-sensitive analyses!
  - But note, not flow- or path-sensitive

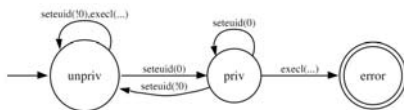
**No Can Do**

- Tom Reps proved this is impossible
- Intuition
  - Set constraints can encode a stack
    - c.f., context sensitive analysis
  - Solving two such problems simultaneously is equivalent to having two stacks
  - Two-stack machines = Turing machines
- Reps uses a reduction to PCP

**What Is Possible?**

- One context-free property
- And any number of regular properties
- Because the intersection of a CFL and a regular language is still context-free

**Example**



**Some Code**

```

s1: setuid(0);           // acquire privilege
s2: if (...) {
s3:   setuid(getuid()); // drop privilege
    }
    else {
s4: ...
    }
s5: execl("/bin/sh", "sh", NULL);
s6: ...
    
```

### Regular Extension to Set Constraints

- Constructors can be annotated with a word from a regular language

$$c^w(t_1, \dots, t_n)$$

- Append operation

$$c^w(t_1, \dots, t_n) \circ w' = c^w \circ w'(t_1 \circ w', \dots, t_n \circ w')$$

### Constraints

$$X \subseteq w' Y$$

$$c^w(t_1, \dots, t_n) \in X \implies c^w(t_1, \dots, t_n) \circ w' \in Y$$

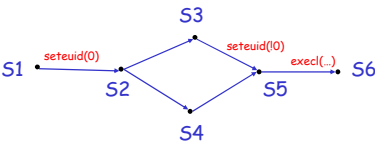
### Example

s1: seteuid(0);	S1 $\sqsubseteq_{\text{seteuid}(0)}$ S2
s2: if (...) {	S2 $\sqsubseteq$ S3
s3: seteuid(getuid());	S2 $\sqsubseteq$ S4
}	S3 $\sqsubseteq_{\text{seteuid}(0)}$ S5
else {	S4 $\sqsubseteq$ S5
s4: ...	S5 $\sqsubseteq_{\text{execl}(...)}$ S6
}	
s5: execl("/bin/sh", "sh", NULL);	
s6: ...	

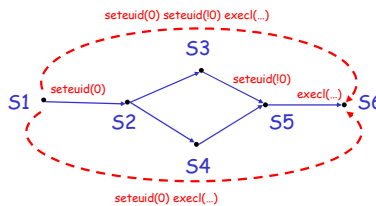
### Example

s1: seteuid(0);	S1 $\sqsubseteq_{\text{seteuid}(0)}$ S2
s2: if (...) {	S2 $\sqsubseteq$ S3
s3: seteuid(getuid());	S2 $\sqsubseteq$ S4
}	S3 $\sqsubseteq_{\text{seteuid}(0)}$ S5
else {	S4 $\sqsubseteq$ S5
s4: ...	S5 $\sqsubseteq_{\text{execl}(...)}$ S6
}	
s5: execl("/bin/sh", "sh", NULL);	
s6: ...	

### Constraint Graph



### Constraint Graph



### Problem

---

- With this representation, there is no bound on the number of edges between nodes in the solved form
  - Every edge may be labeled by a word
  - And there are an infinite number of words

Prof. Aiken & Dill CS 357 Lecture 20

43

### Solution

---

- Every word defines a function from states to states of the DFA
  - Finite # of states  $\rightarrow$  finite # of functions
- Algorithm
  - Replace each word by its state function
  - Use the rule
 
$$X \subseteq f^1 Y \subseteq f^2 Z \Rightarrow X \subseteq f^{2of1} Z$$
  - Finite # of functions  $\rightarrow$  finite number of edges in the closed graph

Prof. Aiken & Dill CS 357 Lecture 20

44

### Complexity & Performance

---

- Cubic in the number of expressions (nodes)
- Linear in the number of state  $\rightarrow$  state functions
  - But this can be exponential in the size of the DFA!
  - See paper for differences between forward, backward, and bidirectional solving
- In practice, faster than hand-written push-down model checking implementations

Prof. Aiken & Dill CS 357 Lecture 20

45

### Context Sensitivity, Reprise

---

- CFL-reachability is one approach to context-sensitivity
- Popular alternative
  - Collapse cycles in the call graph
  - Use resulting finite stacks to distinguish contexts
  - i.e., make the contexts regular
- CFL-reachability is more general, but is there a benefit in practice?
  - My opinion: probably, at least for some problems, but no one really knows

Prof. Aiken & Dill CS 357 Lecture 20

46

### Final Thoughts

---

- Still a big gulf between analysis communities
  - Terminology
  - Technical emphasis
  - Applications

Prof. Aiken & Dill CS 357 Lecture 20

47