

Correctness, Continued

17-654/17-764
Analysis of Software Artifacts
Jonathan Aldrich



Analysis of Software Artifacts -
Spring 2006

Nonterminating Analysis



	Iter	Position	x	y
(bad) idea: Track set of values for each variable	1	0	Z	Z
	2	1	{0}	Z
	3	2	{0}	Z
	4	3	{1}	Z
	5	2	{0,1}	Z
$[x := 0]_1$	6	3	{1,2}	Z
while $[x < y]_2$ do	7	2	{0,1,2}	Z
$[x := x + 1]_3$;	8	3	{1,2,3}	Z
	9	2	{0,1,2,3}	Z
$[x := 0]_4$;	10	3	{1,2,3,4}	Z
	...			

Moral: make your lattices finite height!

Analysis of Software Artifacts -
Spring 2006

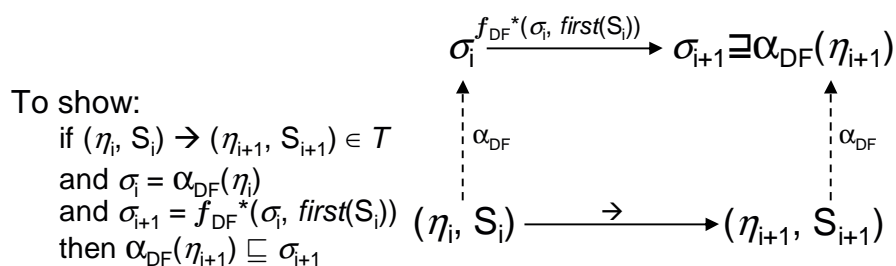
2

Review: What does Correctness Mean?



- Intuition
 - At a fixed point, analysis results are a *sound 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, $\alpha(T(t)) \sqsubseteq \sigma_{pp(T(t))}$ where $\sigma_{pp(T(t))}$ is the analysis results at $pp(T(t))$

Review: Local Soundness



Intuitively, translating from concrete to abstract and applying the flow function will safely approximate (\sqsupseteq) taking a step in the trace and translating from concrete to abstract

Transitive flow function f_{DF}^*



- Applies f_{DF} to an entire statement, rather than to only one AST node

$$f_{DF}^*(\sigma, x := a) = f_{DF}(f_{DF}^*(\sigma, a), x := a)$$

$$f_{DF}^*(\sigma, x) = f_{DF}(\sigma, x)$$

$$f_{DF}^*(\sigma, n) = f_{DF}(\sigma, n)$$

$$f_{DF}^*(\sigma, a_1 \text{ op}_a a_2) = f_{DF}(f_{DF}^*(f_{DF}^*(\sigma, a_1), a_2), a_1 \text{ op}_a a_2)$$

Finding Errors with Local Soundness



- Consider the **incorrect** flow function:
$$f_{ZA}(\sigma, [[\dots]_n + [\dots]_m]_k) =$$
$$\text{if } \sigma[t_n]=Z \parallel \sigma[t_m]=Z$$
$$\text{then } [t_k \mapsto Z]\sigma \text{ else } [t_k \mapsto MZ]\sigma$$
- Local Soundness fails!
 - Consider $\eta_i = []$, $S_i = [x := 3+0]_k$
 - $\sigma_i = \alpha_{DF}(\eta_i) = \alpha_{DF}([]) = []$
 - $\sigma_{i+1} = f_{DF}^*(\sigma_i, \text{first}(S_i)) = [x \mapsto Z]$
 - $\alpha_{DF}(\eta_{i+1}) = \alpha_{DF}([x \mapsto 3]) = [x \mapsto NZ]$
 - $\alpha_{DF}(\eta_{i+1}) \not\sqsubseteq \sigma_{i+1}$ because $Z \not\sqsubseteq NZ$

Local Soundness for Zero Analysis



To show:

if $(\eta_i, S_i) \rightarrow (\eta_{i+1}, S_{i+1}) \in T$
 and $\sigma_i = \alpha_{DF}(\eta_i)$
 and $\sigma_{i+1} = f_{DF}^*(\sigma_i, first(S_i))$
 then $\alpha_{DF}(\eta_{i+1}) \sqsubseteq \sigma_{i+1}$
 (ignoring the temporaries in σ_{i+1})

By induction on the derivation of
 $(\eta_i, S_i) \rightarrow (\eta_{i+1}, S_{i+1})$

Example case:

$$\frac{\eta \vdash a \downarrow v}{(\eta, x := a) \mapsto (\eta[x \mapsto v], skip)}$$

- $\sigma_{i+1} = f_{ZA}^*(\sigma_i, [x := [a]_n]_k)$
 $= f_{DF}(f_{DF}^*(\sigma_i, [a]_n), [x := [a]_n]_k)$
 $= \text{let } \sigma' = f_{DF}^*(\sigma_i, [a]_n)$
 in $[x \mapsto \sigma'(t_n)] \sigma'$
- $\alpha_{ZA}(\eta_{i+1}) = \alpha_{ZA}([x \mapsto v] \eta_i)$
 $= [x \mapsto \alpha_{ZA}(v)] \alpha_{ZA}(\eta_i)$
 $= [x \mapsto \alpha_{ZA}(v)] \sigma_i$
- Lemma: if $\eta_i \vdash [a]_n \downarrow v$
 and $\sigma_i = \alpha_{ZA}(\eta_i)$
 and $\sigma' = f_{ZA}^*(\sigma_i, [a]_n)$
 then $\alpha_{ZI}(v) \sqsubseteq \sigma'(t_n)$
- Lemma: $f_{DF}^*(\sigma_i, [a]_n)$ affects only temporaries
- By Lemma, we can conclude that
 $[x \mapsto \alpha_{ZA}(v)] \sigma_i \sqsubseteq [x \mapsto \sigma'(t_n)] \sigma'$, and so the
 case holds

Local Soundness for Zero Analysis



Lemma: if $\eta_i \vdash [a]_k \downarrow v$
 and $\sigma_i = \alpha_{ZA}(\eta_i)$
 and $\sigma' = f_{ZA}^*(\sigma_i, [a]_k)$
 then $\alpha_{ZI}(v) \sqsubseteq \sigma'(t_k)$

By induction on the derivation of
 $\eta_i \vdash [a]_k \downarrow v$

Example case:

$$\eta \vdash n \downarrow n$$

- $\sigma' = f_{ZA}^*(\sigma_i, [n]_k)$
 $= \text{if } n == 0 \text{ then } [t_k \mapsto Z] \sigma_i$
 else $[t_k \mapsto NZ] \sigma_i$
- $\alpha_{ZI}(v) = \alpha_{ZI}(n)$
 $= \text{if } n == 0 \text{ then } Z$
 else NZ
- Case $n == 0$:
 - $\sigma'(t_k) = [t_k \mapsto Z] \sigma_i(t_k) = Z$,
 - $\alpha_{ZI}(v) = Z$
 - so $\alpha_{ZI}(v) \sqsubseteq \sigma'(t_k)$
- Case $n != 0$:
 - $\sigma'(t_k) = [t_k \mapsto NZ] \sigma_i(t_k) = NZ$
 - $\alpha_{ZI}(v) = NZ$
 - so $\alpha_{ZI}(v) \sqsubseteq \sigma'(t_k)$

The Proof Fails!



Lemma: if $\eta_i \vdash [a]_k \downarrow v$
 and $\sigma_i = \alpha_{ZA}(\eta_i)$
 and $\sigma = f_{ZA}^*(\sigma_i, [a]_k)$
 then $\alpha_{Zl}(v) \sqsubseteq \sigma'(t_k)$

By induction on the derivation of
 $\eta_i \vdash [a]_n \downarrow v$

Example case:

$$\frac{\eta \vdash a \downarrow v \quad \eta \vdash a' \downarrow v'}{\eta \vdash a \text{ op } a' \downarrow v \text{ op } v'}$$

Evaluate for op = +

- $\sigma = f_{ZA}^*(\sigma_i, [[a]_l]_n + [a_2]_m)_k$
 $= \text{let } \sigma' = f_{DF}^*(f_{DF}^*(\sigma', [a]_l)_n, [a_2]_m)$
 in if $\sigma'[t_n]=Z \parallel \sigma'[t_m]=Z$
 then $[t_k \mapsto Z]\sigma'$ else $[t_k \mapsto MZ]\sigma'$
- $\alpha_{Zl}(v + v')$
 $= \text{if } (v + v') == 0 \text{ then } Z$
 else NZ
- By I.H. $\alpha_{Zl}(v) \sqsubseteq f_{DF}^*(\sigma', [a]_l)_n(t_n)$
- By I.H. $\alpha_{Zl}(v') \sqsubseteq f_{DF}^*(f_{DF}^*(\sigma', [a]_l)_n, [a_2]_m)(t_m)$
 - Also need lemma regarding temporaries
- If $v=0$ or $v'=0$ then $\sigma'[t_n]=Z$ or MZ and $\sigma'[t_m]=Z$ or MZ, so $\sigma'(t_k) = Z$ or MZ. But $\alpha_{Zl}(v + v') = Z$ or NZ.
- We cannot eliminate the possibility that $\sigma'(t_k) = Z$ and $\alpha_{Zl}(v + v') = NZ$, in which case $\alpha_{Zl}(v) \not\sqsubseteq \sigma'(t_k)$ and the lemma fails.

Proving the Corrected Analysis



Lemma: if $\eta_i \vdash [a]_k \downarrow v$
 and $\sigma_i = \alpha_{ZA}(\eta_i)$
 and $\sigma = f_{ZA}^*(\sigma_i, [a]_k)$
 then $\alpha_{Zl}(v) \sqsubseteq \sigma'(t_k)$

By induction on the derivation of
 $\eta_i \vdash [a]_n \downarrow v$

Example case:

$$\frac{\eta \vdash a \downarrow v \quad \eta \vdash a' \downarrow v'}{\eta \vdash a \text{ op } a' \downarrow v \text{ op } v'}$$

Evaluate for op = +

- $\sigma = f_{ZA}^*(\sigma_i, [[a]_l]_n + [a_2]_m)_k$
 $= \text{let } \sigma' = f_{DF}^*(f_{DF}^*(\sigma', [a]_l)_n, [a_2]_m)$
 in if $\sigma'[t_n]=Z \ \&\& \ \sigma'[t_m]=Z$
 then $[t_k \mapsto Z]\sigma'$ else $[t_k \mapsto MZ]\sigma'$
- $\alpha_{Zl}(v + v')$
 $= \text{if } (v + v') == 0 \text{ then } Z$
 else NZ
- By I.H. $\alpha_{Zl}(v) \sqsubseteq f_{DF}^*(\sigma', [a]_l)_n(t_n)$
- By I.H. $\alpha_{Zl}(v') \sqsubseteq f_{DF}^*(f_{DF}^*(\sigma', [a]_l)_n, [a_2]_m)(t_m)$
 - Also need lemma regarding temporaries
- If $v=0$ and $v'=0$ then $\sigma'[t_n]=Z$ or MZ and $\sigma'[t_m]=Z$ or MZ, so $\sigma'(t_k) = Z$ or MZ. Now $\alpha_{Zl}(v + v') = Z$ so either way $\alpha_{Zl}(v) \sqsubseteq \sigma'(t_k)$.
- Otherwise either $v \neq 0$ and $v' = 0$, so either $\sigma'[t_n]=MZ$ or $\sigma'[t_m]=MZ$, so $\sigma'(t_k) = MZ$. Since MZ is the top of the lattice, $\alpha_{Zl}(v) \sqsubseteq \sigma'(t_k)$.

Global Soundness



- Intuition
 - We begin with initial dataflow facts ι that safely approximate (\sqsupseteq) all initial stores η_0
 - By local soundness, each transfer function when given safe input information yields safe output information
 - By induction, any fixed point of the analysis is sound

Global Soundness



- Theorem (Global Soundness)
 - Assume that $\forall T \in \text{traces}(S) \alpha_{\text{DF}}(\eta_0) \sqsupseteq \iota$ and that analysis DF is monotone and locally sound with respect to α_{DF}
 - Then for any fixed point DF_{fix} of DF on program S, $\forall T \in \text{traces}(S) \forall t \in \text{times}(T)$ we have $\alpha_{\text{DF}}(\eta_t) \sqsupseteq DF_{\text{fix}}(\text{pp}(T(t)))$
- Proof outline: For each trace T we do induction on t
 - Induction hypothesis: $\alpha_{\text{DF}}(\eta_t) \sqsupseteq DF_{\text{fix}}(\text{pp}(T(t)))$
 - Base case: $t=0$
 - By assumption $\alpha_{\text{DF}}(\eta_0) \sqsupseteq \iota = DF_{\text{fix}}(\text{pp}(\eta_0))$
 - Inductive case: time t and statement S_t
 - *Simplifying assumption: straight-line control flow*
 - By induction hypothesis we have $\alpha_{\text{DF}}(\eta_{t-1}) \sqsupseteq DF_{\text{fix}}(\text{pp}(T(t-1)))$
 - By monotonicity of DF we have:
 $f_{\text{DF}}(\alpha_{\text{DF}}(\eta_{t-1}), S_t) \sqsupseteq f_{\text{DF}}(DF_{\text{fix}}(\text{pp}(T(t-1))), S_t)$
 - By local soundness we have $\alpha_{\text{DF}}(\eta_t) \sqsupseteq f_{\text{DF}}(\alpha_{\text{DF}}(\eta_{t-1}), S_t)$
 - By transitivity we get $\alpha_{\text{DF}}(\eta_t) \sqsupseteq f_{\text{DF}}(DF_{\text{fix}}(\text{pp}(T(t-1))), S_t)$
 - But $f_{\text{DF}}(DF_{\text{fix}}(\text{pp}(T(t-1))), S_t) = DF_{\text{fix}}(\text{pp}(T(t)))$ because it's a fixed point
 - So we have $\alpha_{\text{DF}}(\eta_t) \sqsupseteq DF_{\text{fix}}(\text{pp}(T(t)))$

PREfix

Reading: **A Static Analyzer for Finding Dynamic Programming Errors**

17-654/17-765

Analysis of Software Artifacts

Jonathan Aldrich



Analysis of Software Artifacts -
Spring 2006

Find the Bugs!



```
char *f(int size) {  
    char * result;  
    if (size > 0)  
        result = (char *)malloc(size);  
    if (size == 1)  
        return NULL;           // memory leak  
    result[0] = 0;             // result may be uninitialized  
                                // malloc may have failed  
    return result;  
}
```

Analysis of Software Artifacts -
Spring 2006

15

Motivation



- Finding programming errors
 - invalid pointers
 - storage allocation errors
 - uninitialized memory
 - improper operations on resources

Can't we just test?



- 90% of errors involve interactions of multiple functions
 - Is this why the original developer didn't find them?
- Occur in unusual or error conditions
 - Often hard to exercise with testing

Problems with Other Tools



- **False Negatives**
 - They look only in one function and miss errors
- **False Positives**
 - They report errors that can't really occur
- **Hard to use**
 - Require extensive program annotations
- **Require test cases**
 - May be impractical
 - Only as good as your test suite

Goals of PREFIX



- **Handle hard aspects of C-like languages**
 - Pointers, arrays, unions, libraries, casts...
- **Don't require user annotations**
 - Build on language semantics
- **Avoid false positives**
 - Use path-sensitive analysis
- **Give the user good feedback**
 - Why might an error occur? Show the user an example execution

PREfix Analysis



- Explore paths through function
- For each path:
 - Symbolically execute path
 - Determine facts true along the path
 - Compute a guard
 - What must be true for the path to be taken
 - Compute constraints
 - Preconditions for successful execution of path
 - Compute result
 - What is true of the return value?

PREfix: Analysis Example

(syntax slightly de-LISP-ified)



```
char *f(int size) {
  char * ptr;
  if (size > 0)
    ptr=(char*)malloc(size);
  if (size == 1)
    return NULL;
  ptr[0] = 0;
  return ptr;
}

f (param size)
alternate 0
  guard size <= 0
  constraint initialized(size)
  ARRAY ACCESS ERROR: ptr not initialized
alternate 1
  guard size == 1
  constraint initialized(size)
  fact ptr==memory_new(size)
  result return==NULL
  MEMORY LEAK ERROR:
  memory pointed to by ptr is not reachable
  through externally visible state
alternate 2
  guard size > 1
  constraint initialized(size)
  fact ptr==NULL
  ARRAY ACCESS ERROR: ptr is NULL
alternate 3
  guard size > 1
  constraint initialized(size)
  fact ptr==memory_new(size)
  fact ptr[0] == 0
  result return == memory_new(size) && return[0] == 0
alternate 4...
```

Big Ideas



- Path sensitivity
 - Avoids reporting errors that occur on control-flow paths that can't really be taken
- Dynamic analysis
 - Explores a *subset* of possible program executions
 - May not find all errors, but still useful
 - Carefully constructed to cover more functionality than most testing strategies can
- Interprocedural analysis
 - Looks at how the behavior of a callee affects the caller

Motivation: Path Sensitivity



$[z := 0]_1$

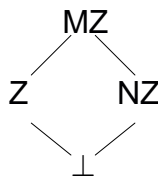
if $[b]_2$

$[z := 10]_3;$

$[x := 100]_4;$

if $[b]_5$

$[x := x / z]_6;$

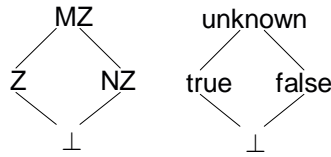


	<u>after pp</u>	<u>z</u>
0		MZ
1		Z
2		Z
3		NZ
4		MZ
5		MZ

**Warning: possible
divide by zero**

- Does this code have a bug?
- What would zero analysis say?

Path Sensitive Analysis



```

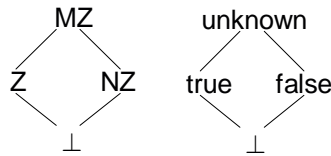
[z := 0]1
if [b]2
  [z := 10]3;
[x := 100]4;
if [b]5
  [x := x / z]6;
  
```

Analysis value after statement

- 0: [z→MZ, b→unknown]
- 1: [z→Z, b→unknown]
- 2: Split path into p and q on b
- 2p: [z→Z, b→true], take branch
- 3p: [z→NZ, b→true]
- 4p: [z→NZ, x→NZ, b→true]
- 5p: [z→NZ, x→NZ, b→true], take branch
- 6p: [z→NZ, x→NZ, b→true]

No error

Path Sensitive Analysis



```

[z := 0]1
if [b]2
  [z := 10]3;
[x := 100]4;
if [b]5
  [x := x / z]6;
  
```

Analysis value after statement

- 0: [z→MZ, b→unknown]
- 1: [z→Z, b→unknown]
- 2: Split path into p and q on b
- 2q: [z→Z, b→false], skip branch
- 4q: [z→Z, x→NZ, b→false]
- 5q: [z→Z, x→NZ, b→false], skip branch

No error

Path-Sensitive Analysis



Analyzes each feasible program path separately

- Benefit
 - Increased precision from eliminating infeasible paths
- Cost
 - Exponential number of paths
- Loops
 - Infinite number of paths—cannot explore them all

Path Sensitivity: Addressing the Cost



- How does PREFIX deal with
 - Exponential path blowup?
 - Explore up to a fixed number of paths
 - Merge paths with identical results
 - Loops?
 - Explore up to a fixed number of iterations

What if you miss a path?



```
char *f(int size) {
  char * ptr;
  if (size > 0)
    ptr=(char*)malloc(size);
  if (size == 1)
    return NULL;
  ptr[0] = 0;
  return ptr;
}

f (param size)
alternate 0
  guard size <= 0
  constraint initialized(size)
  ARRAY ACCESS ERROR: ptr not initialized
alternate 1
  guard size == 1
  constraint initialized(size)
  fact ptr==memory_new(size)
  result return==NULL
  MEMORY LEAK ERROR:
  memory pointed to by ptr is not reachable
  through externally visible state
alternate 2
  guard size > 1
  constraint initialized(size)
  fact ptr==NULL
  ARRAY ACCESS ERROR: ptr is NULL
alternate 3
  guard size > 1
  constraint initialized(size)
  fact ptr==memory_new(size)
  fact ptr[0] == 0
  result return == memory_new(size) && return[0] == 0
alternate 4...
```

Soundness for PREFIX



- Exploring only some paths is unsound
 - Might miss bugs on paths not explored
- Sound alternatives
 - Explore a fixed set of paths/iterations
 - Merge all other paths together using dataflow analysis to reach a fixed point
 - Cost
 - May yield too many false positive error reports
 - PREFIX chooses unsoundness to avoid false positives