Interprocedural Analysis

17-654/17-754
Analysis of Software Artifacts
Jonathan Aldrich

Interprocedural Analysis Strategies

- Make default assumptions
- Assume and check annotations
- Build Interprocedural CFG
- Compute Summaries
  - All contexts one by one
  - Each context on demand
  - All context at once

© 2008 Jonathan Aldrich

Making Default Assumptions

- Assumptions
  - Starting dataflow value for all parameters
  - Dataflow value for result
  - Starting and ending information for globals if you're tracking them (*most analyses don't*)
- Verification
  - Initial info: starting value for parameters
  - Verify result $\sqsubseteq$ $\text{assumption}_{result}$
    - Ending value for result obeys assumption
  - Verify arg $\sqsubseteq$ assumption_{arg}
    - Actual arguments obey assumptions of formal parameter
  - Similar extension to globals if you're tracking them

29 February 2008 | Analysis of Software Artifacts | 3

© 2008 Jonathan Aldrich

Making Default Assumptions

- Example: Zero Analysis
  - Default: $\top$ (MZ) for arguments and results
    - Benefit: actual arguments and actual result always obey assumption
    - Cost: very conservative for arguments
      - Will report false positive errors
  - Globals: easiest solution is not to track them
    - Could also track but assume $\top$ at boundaries

29 February 2008 | Analysis of Software Artifacts | 4

© 2008 Jonathan Aldrich

Example: Default Assumptions

```
int divByX(int x) {
    [result := 10/x]1;
}

void caller() {
    [x := 5]1;
    [y := divByX(x)]2;
}
```

- Analyze divByX

p	x	result
0	MZ	MZ
1	MZ	NZ

  - Warning: div by zero at 1
  - Verify σ[result] ⊑ MZ
- Analyze caller

p	x	y
0	MZ	MZ
1	NZ	MZ
2	NZ	MZ

  - Verify σ[x] ⊑ MZ
- Note that div by zero can't happen!

Optimistic Assumption: NZ

```
int divByX(int x) {
    [result := 10/x]1;
}

void caller() {
    [x := 5]1;
    [y := divByX(x)]2;
}
```

- Analyze divByX

p	x	result
0	NZ	MZ
1	NZ	NZ

  - No warning
  - Verify σ[result] ⊑ NZ
- Analyze caller

p	x	y
0	MZ	MZ
1	NZ	MZ
2	NZ	NZ

  - Verify σ[x] ⊑ NZ

Optimistic Assumption: NZ

```
int double(int x) {
    [result := 2*x]1;
}

void caller() {
    [x := 0]1;
    [y := double(x)]2;
}
```

- Analyze double

p	x	result
0	NZ	MZ
1	NZ	NZ

  - No warning
  - Verify σ[result] ⊑ NZ
- Analyze caller

p	x	y
0	MZ	MZ
1	Z	MZ
2	Z	NZ

  - Verify σ[x] ⊑ NZ fails!
  - False positive—this code is OK

© 2008 Jonathan Aldrich

Assume and Check Annotations

- Annotations
  - Starting dataflow value for all parameters
  - Dataflow value for result
- Verification
  - Initial info: starting value for parameters
  - Verify result ⊑ $\text{annotation}_{result}$
    - Ending value for result obeys annotation
  - Verify arg ⊑ annotation_{arg}
    - Actual arguments obey annotations on formal parameter

© 2008 Jonathan Aldrich

Assumption Example

```
@NZ int divByX(@NZ int x) {
    [result := 10/x]1;
}

void caller() {
    [x := 5]1;
    [y := divByX(x)]2;
}
```

- Analyze divByX

p	x	result
0	NZ	MZ
1	NZ	NZ

  - Verify σ[result] ⊑ NZ
- Analyze caller

p	x	y
0	MZ	MZ
1	NZ	MZ
2	NZ	NZ

  - Verify σ[x] ⊑ NZ

© 2008 Jonathan Aldrich

Assumption Example

```
@MZ int double(@MZ int x) {
    [result := 2*x]1;
}

void caller() {
    [x := 5]1;
    [y := double(x)]2;
    [z := 10/y]3;
}
```

- Analyze divByX

p	x	result
0	MZ	MZ
1	MZ	MZ

  - Verify σ[result] ⊑ MZ
- Analyze caller

p	x	y	z
0	MZ	MZ	MZ
1	NZ	MZ	MZ
2	NZ	MZ	MZ

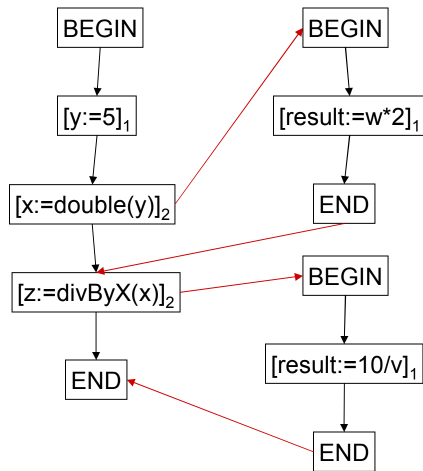
  - Verify σ[x] ⊑ MZ

p	x	y	z
3	NZ	MZ	MZ

  - Warning: possible div by zero
  - False positive!

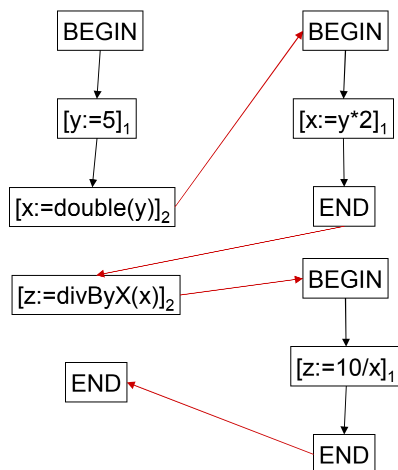
© 2008 Jonathan Aldrich

Interprocedural CFG Intuition



© 2008 Jonathan Aldrich

Interprocedural CFG Intuition



© 2008 Jonathan Aldrich

Interprocedural CFG Example

```
int double(int x) {
    [y := 2*x]6;
}

void caller() {
    [x := 5]1;
    [y := double(x)]2;
    [z := 10/y]3;
    [x := 0]4;
    [y := double(x)]5;
}
```

p	x	y	z
0	MZ	MZ	MZ
1	NZ	MZ	MZ
6	NZ	NZ	MZ
2	NZ	NZ	MZ
3	NZ	NZ	NZ

- No div by zero

p	x	y	z
4	Z	NZ	NZ
6	MZ	MZ	MZ
5	MZ	MZ	MZ

- Must revisit node 2 because result of double changed

p	x	y	z
2	MZ	MZ	MZ
3	MZ	MZ	NZ

- Divide by zero warning
- False positive!

© 2008 Jonathan Aldrich

Context Sensitive Summaries

- Intuition
  - Interprocedural CFG loses too much precision when a function is called with different argument dataflow lattice elements
  - Simple annotations have same issue
    - (but same Summary technique works there)
- Summaries
  - Maps from input dataflow information to output dataflow information
  - *Context sensitive*: different results for different calls
  - When function is called, apply the map!

© 2008 Jonathan Aldrich

Generating Context Sensitive Summaries

- Brute force
  - Analyze the function once for each possible input lattice element
  - Problem: way too many lattice elements—would take too long
- On demand
  - Analyze the function once for each actual input lattice element it is called with
  - Much better—but can still be impractical for large programs with precise lattices
- Abstract summaries
  - Symbolically represent function's effect on input lattice element
  - Example: PREfix's technique
  - The state of the art in interprocedural analysis

© 2008 Jonathan Aldrich

On Demand Summaries

```
/* Summary
 * Case x:NZ -> result:NZ

*/
int double(int x) {
   [result := 2*x]1;
}

void caller() {
   [x := 5]1;
   [y := double(x)]2;
   [z := 10/y]3;
   [x := 0]4;
   [y := double(x)]5;
}
```

p	x	y	z
0	MZ	MZ	MZ
1	NZ	MZ	MZ
2	NZ	NZ	MZ

Compute summary of double for x:NZ

p	x	result
0	NZ	MZ
1	NZ	NZ

© 2008 Jonathan Aldrich

On Demand Summaries

```
/* Summary
 * Case x:NZ -> result:NZ
 * Case x:Z -> result:Z
*/
int double(int x) {
   [result := 2*x]1;
}

void caller() {
   [x := 5]1;
   [y := double(x)]2;
   [z := 10/y]3;
   [x := 0]4;
   [y := double(x)]5;
}
```

p	x	y	z
0	MZ	MZ	MZ
1	NZ	MZ	MZ
2	NZ	NZ	MZ
3	NZ	NZ	NZ
4	Z	NZ	NZ
5	Z	Z	NZ

Compute summary of double for x:Z

p	x	result
0	Z	MZ
1	Z	Z

© 2008 Jonathan Aldrich

Context Sensitive Annotations

```
@Case("x:NZ -> result:NZ")
@Case("x:Z -> result:Z")
int double(int x) {
   [result := 2*x]1;
}

void caller() {
   [x := 5]1;
   [y := double(x)]2;
   [z := 10/y]3;
   [x := 0]4;
   [y := double(x)]5;
}
```

Verify annotation @Case("x:Z -> result:Z")

p	x	result
0	Z	MZ
1	Z	Z

Verify annotation @Case("x:NZ -> result:NZ")

p	x	result
0	NZ	MZ
1	NZ	NZ

Verify client

p	x	y	z
0	MZ	MZ	MZ
1	NZ	MZ	MZ
2	NZ	NZ	MZ *// case x:NZ*
3	NZ	NZ	NZ
4	Z	NZ	NZ
5	Z	Z	NZ *// case x:Z*

© 2008 Jonathan Aldrich

Abstract Summaries

```
/* Summary
 * Case x:α -> result:α
 */
int double(int x) {
    [result := 2*x]1;
}

void caller() {
    [x := 5]1;
    [y := double(x)]2;
    [z := 10/y]3;
    [x := 0]4;
    [y := double(x)]5;
}
```

Compute summary of double for x:α

p	x	result
0	α	MZ
1	α	α

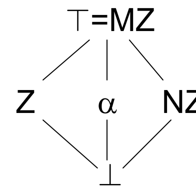
Analyze client

p	x	y	z
0	MZ	MZ	MZ
1	NZ	MZ	MZ
2	NZ	NZ	MZ *// α=NZ*
3	NZ	NZ	NZ
4	Z	NZ	NZ
5	Z	Z	NZ *// α=Z*

© 2008 Jonathan Aldrich

Abstract Summaries for Zero Analysis

- New ZA lattice has α
- Flow functions
  - $f_{ZA}(\sigma, [x]_k) = [t_k \mapsto \sigma(x)]\ \sigma$
  - $f_{ZA}(\sigma, [n]_k) =$ if n==0
    then $[t_k \mapsto Z]\sigma$
    else $[t_k \mapsto NZ]\sigma$
  - $f_{ZA}(\sigma, [x := [\ldots]_n]_k) = [x \mapsto \sigma(t_n)]\ \sigma$
  - $f_{ZA}(\sigma, [[\ldots]_n\ op\ [\ldots]_m]_k) =$
    if op=* and $\sigma[t_m]$=NZ
    then $[t_k \mapsto \sigma(t_n)]\sigma$ *// this case used in example*
    if …
    else $[t_k \mapsto MZ]\ \sigma$
  - $f_{ZA}(\sigma,$ */* any other */*$) = \sigma$
- Many other ways to generate summaries



© 2008 Jonathan Aldrich

Comparison

- Assumptions
  - Simple, efficient
  - Imprecise
- Annotations
  - Require effort
  - More precise than assumptions
  - More efficient than IP analysis
  - Can used “summary annotations” to get context sensitivity
- Both work on partial programs
- Interprocedural CFG
  - Simple for programmer
  - As precise as simple annotations
  - Still imprecise, can be very costly
    - $O(n^3)$ in size of program
- Summaries
  - Excellent precision
  - Costly if not abstract
- Both require whole program

© 2008 Jonathan Aldrich