

Context-Sensitive Pointer Analysis

Last time

- Flow-insensitive pointer analysis

Today

- Context-sensitive pointer analysis
 - Emami invocation graphs
 - Partial Transfer Functions
- The big picture

Recall Context Sensitivity

Is x constant?

```

a = id(4);      b = id(5);
      ↙ ↘      ↙ ↘
      4 5      4 5
      ↘ ↙      ↘ ↙
      id(x) { return x; }
    
```

Context-sensitive analysis

- Computes an answer for every callsite:
 - **x** is 4 in the first call
 - **x** is 5 in the second call

```

a = id(4);      b = id(5);
      ↙ ↘      ↙ ↘
      4,5      4,5
      ↘ ↙      ↘ ↙
      id(x) { return x; }
    
```

Emami 1994

Overview

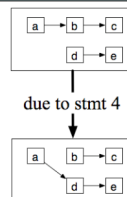
- Uses invocation graph for context-sensitivity
- Can be exponential in program size
- Handles function pointers

Characterization of Emami

- Whole program
- Flow-sensitive
- Context-sensitive
- May and must analysis
- Alias representation: points-to
- Heap modeling: one heap variable
- Aggregate modeling of fields and arrays

```

int **a, *b, c, *d, e;
1: a = &b;
2: b = &c;
3: d = &e;
4: a = &d;
    
```



Partial Transfer Functions [Wilson et. al. 95]

Key idea

- Exploit commonality among contexts
- Provide one procedure summary (PTF) for all contexts that share the same input/output aliasing relationships

Partial Transfer Functions – Example

```

main() {
  int *a,*b,c,d;
  a = &c;
  b = &d;
  swap(&a, &b); // S0
  for (i = 0; i<2; i++) {
    bar(&a,&a); // S1
    bar(&b,&b); // S2
    bar(&a,&b); // S3
    bar(&b,&a); // S4
  }
}
void bar(int **i, int **j) { swap(i,j); }
void swap(int **x, int **y) {
  int *temp = *x;
  *x = *y;
  *y = temp;
}

```

How many contexts do we care about?

- Two: the formals either alias or they do not alias

In practice

- Only need 1 or 2 PTF's per procedure
- Complex to implement

6

The Big Picture

Where do we lose precision?

- Let's revisit our running example from last week

CS553 Lecture

Context-Sensitive Pointer Analysis

7

Revisiting Our Earlier Example

Flow-insensitive context-sensitive (FICS)

```

int** foo(int **p, **q)
{
  int **x;           p1 →
  x = p;             p2 →
  . . .              q1 →
  x = q;             q2 →
  return x;          x1 →
}
int main()
{
  int **a, *b, *d, *f,  a →
    c, e;               b →
  a = foo(&b, &f);      d →
  *a = &c;              f →
  a = foo(&d, &g);      g →
  *a = &e;
}

```

CS553 Lecture

Context-Sensitive Pointer Analysis

8

Revisiting Our Earlier Example (cont)

Flow-sensitive context-sensitive (FSCS)

```

int** foo(int **p, **q)
{
  int **x;           P11 →
  x = p;             q11 →
  . . .              p21 →
  x = q;             q21 →
  return x;          x11 →
}
int main()
{
  int **a, *b, *d, *f,  x21 →
    c, e;               x22 →
  a = foo(&b, &f);      a11 →
  *a = &c;              a12 →
  a = foo(&d, &g);      f11 →
  *a = &e;              g11 →
}

```

first callsite
first def

CS553 Lecture

Context-Sensitive Pointer Analysis

9

Revisiting Our Earlier Example (cont)

Flow-insensitive context-insensitive (FICI)

```

int** foo(int **p, **q)
{
    int **x;

    x = p;
    . . .
    x = q;
    return x;
}

int main()
{
    int **a, *b, *d, *f,
        c, e;

    a = foo(&b, &f);
    *a = &c;
    a = foo(&d, &g);
    *a = &e;
}

```

p → {b, d}
 q → {f, g}
 x → {b, d, f, g}
 a → {b, d, f, g}
 b → {c, e}
 d → {c, e}
 f → {c, e}
 g → {c, e}

CS553 Lecture

Context-Sensitive Pointer Analysis

10

Revisiting Our Earlier Example (cont)

Flow-sensitive context-insensitive (FSCI)

```

int** foo(int **p, **q)
{
    int **x;

    x = p;
    . . .
    x = q;
    return x;
}

int main()
{
    int **a, *b, *d, *f,
        c, e;

    a = foo(&b, &f);
    *a = &c;
    a = foo(&d, &g);
    *a = &e;
}

```

p → {b, d}
 q → {f, g}
 x₁ → {b, d}
 x₂ → {f, g}
 a₁ → {f, g}
 a₂ → {f, g}
 f₁ → {c}
 g₁ → {c}
 f₂ → {c, e} (weak update)
 g₂ → {c, e} (weak update)

CS553 Lecture

Context-Sensitive Pointer Analysis

11

Strong vs. Weak Updates

Strong update

- When we know precisely what an assignment through a pointer refers to, the assignment kills old information
- Such cases are analogous to assignments to scalars

```

int a;
a := 5;
{a=5}
a := 6;
{a=6}

```

```

int *a, b;
a := &b;
{a->b}
b := 5;
{a->b, b=5}
*a := 6;
{a->{b}, b=6}

```

CS553 Lecture

Context-Sensitive Pointer Analysis

12

Strong vs. Weak Updates

Weak update

- When we do not know what an assignment through a pointer refers to, we cannot use that assignment to kill old facts
- So the imprecision spreads

```

int *a, b, c;
if (blah)
    a := &b;
else
    a := &c;
{a->{b,c}}
b := 5;
{a->{b,c}, b=5}
*a := 6;
{a->{b,c}, b=5 □ b=6}

```

Does not kill {b=5} because *a might update c and not b

CS553 Lecture

Context-Sensitive Pointer Analysis

13

Imprecision

Weak updates

- Occur more often in flow-insensitive and context-insensitive analyses

The callgraph

- When function pointers are used, pointer analysis is needed to build the callgraph
- Imprecision in pointer analysis leads to imprecision in the callgraph
 - A conservative callgraph has more edges than a less conservative callgraph
- Imprecision in the callgraph leads to further imprecision in the pointer analysis

The basic issue

- The need for approximation

CS553 Lecture

Context-Sensitive Pointer Analysis

14

Approximations

Many ways to approximate

- Recall that the constraint graph has nodes representing variables and edges representing constraints
- The many dimensions of pointer analysis represent different ways of collapsing the constraint graph

Flow-insensitive

- Andersen:
 - Collapse all constraints (assignments) pertaining to a given variable into a single node
- Steensgaard:
 - Collapse all nodes that have been assigned to one another into a single node
 - Allows information to flow from rhs to lhs as well as from lhs to rhs

CS553 Lecture

Context-Sensitive Pointer Analysis

15

Andersen 94

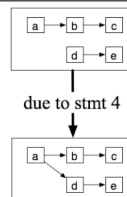
Overview

- Uses subset constraints
- Cubic complexity in program size, $O(n^3)$

Characterization of Andersen

- Whole program
- Flow-insensitive
- Context-insensitive
- May analysis
- Alias representation: points-to
- Heap modeling?
- Aggregate modeling: fields

```
int **a, *b, c, *d, e;  
1: a = &b;  
2: b = &c;  
3: d = &e;  
4: a = &d;
```



source: Barbara Ryder's Reference Analysis slides

CS553 Lecture

Context-Sensitive Pointer Analysis

16

Steensgaard 96

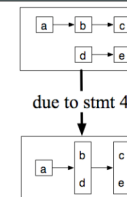
Overview

- Uses unification constraints
- Almost linear in terms of program size
- Uses fast union-find algorithm
- Imprecision from merging points-to sets

Characterization of Steensgaard

- Whole program
- Flow-insensitive
- Context-insensitive
- May analysis
- Alias representation: points-to
- Heap modeling: none
- Aggregate modeling: possibly

```
int **a, *b, c, *d, e;  
1: a = &b;  
2: b = &c;  
3: d = &e;  
4: a = &d;
```



source: Barbara Ryder's Reference Analysis slides

CS553 Lecture

Context-Sensitive Pointer Analysis

17

More Approximations

Context-insensitive analysis

- Collapse all constraints arising from different callsites of a procedure into a single node

Partial Transfer Functions

- Collapse constraints for all callsites of a procedure that share the same aliasing relationships

Field-insensitive

- Collapse all fields of a structure into a single node

Field-based

- Collapse all instances of a struct type into one node per field
- Example: one node for all instances of `student.name`, and another node for all instances of `student.gpa`

CS553 Lecture

Context-Sensitive Pointer Analysis

18

Yet More Approximations

Address Taken

- Collapse all objects that have their address taken into a single node
- Assume that all pointers point to this node

Heap naming

- One heap:
 - Collapse all heap objects into a single node
- Static allocation site
 - Collapse all instances of objects that are allocated at the same program location into a single node

CS553 Lecture

Context-Sensitive Pointer Analysis

19

Concepts

Partial Transfer Functions

- Exploit commonality among contexts

Sources of imprecision

CS553 Lecture

Context-Sensitive Pointer Analysis

20

Next Time

Next lecture

- Profile-guided optimization

CS553 Lecture

Context-Sensitive Pointer Analysis

21