Points-to analysis

Daqing Hou

Outline

- Motivation for points-to analysis
- Steensgaard's algorithm
- Anderson's algorithm
- Summary

Motivation

Points-to is a relation between a pointer/reference variable and the set of locations it points to.

Enable more aggressive optimizations:

- x = 1; *z++; y=x*y; // is x=1 here?
- In particular, in oo, resolve virtual methods o.m(...); // does o resolve to a single class?
- Enable escape analysis and remove redundant synchronizations
- Construct call graphs

Steensgaard's algorithm (1)

Paper: Bjarne Steensgaard. Points-to Analysis in Almost Linear Time. POPL'96

- One points-to set per pointer variable *over entire program*
- for pointer assignment, p = q, makes Pts-to(p)=Pts-to(q) (a.k.a. unification constraints)
  - Uses fast union-find algorithm
  - done recursively for multiple-level pointers
  - reduces the size of the points-to graph (in terms of both nodes and edges)
- Almost linear solution time in terms of program size, O(n)
- Imprecision stems from merging points-to sets

Steensgaard's algorithm (2)

- Find all pointer assignments in program
- Form points-to graph nodes from

pointer variables/fields and variables (in the

heap or whose address has been taken)

1. Examine each statement, *in arbitrary order*, and construct points-to edges
2. Merge nodes (and edges) where indicated by unification constraints

- After linear pass over these assignments, points-to graph is complete

Steensgaard's algorithm (3)

```
1.    a=malloc; // L1
2.    b=malloc; // L2
3.    a=b;
```

```
1.    a=&b;
2.    b=&c;
3.    d=&e;
4.    a=&d;
```

Anderson's algorithm

PhD dissertation: L. Anderson. Program analysis and specialization for the C programming language. University of Copenhagen. May 1994.

- One points-to set per pointer variable over entire program
- For pointer assignment, p = q, makes Pts-to(q) subset of Pts-to(p), a.k.a, inclusion constraints
- Points-to graph larger than Steensgaard's but more precise
- Worst case cubic complexity in program size, O(n3), to construct the points-to graph

Anderson's algorithm

```
1.    a=malloc; // L1
2.    b=malloc; // L2
3.    a=b;
```

```
1.    a=&b;
2.    b=&c;
3.    d=&e;
4.    a=&d;
```

Summary: points-to analysis for C

- Flow- and context-insensitive formulations of points-to analysis

1. Linear scan of assignments; scalable
2. Good enough for ensuring safety of some optimizations & program understanding
3. Not good for applications needing precise def-use information (e.g., ***correctness checking***, program slicing, testing)
4. General approach is unification or inclusion constraints
5. Newer versions kept track of individual struct fields as pointer targets

- Extended to points-to analyses for OOPL reference variables

Acknowledgment

Some slides are customizations of those of Dr. Barbara Ryder of Rutgers U.