The Ant and the Grasshopper: Fast and Accurate pointer analysis for Millions of Lines of Code

Ben Hardekopf, Calvin Lin

University of Texas at Austin

PLDI '07

Introduction

- Points-to analysis is a useful prerequisite for most program analyses.
- Precise pointer analysis is *NP-hard*.
- Most of the precise analyses are flow and context sensitive, but face difficulties in scaling due to their large complexity. Thus, this paper drops both these constraints.
- *Andersen's* analysis is a precise analysis which runs in $O(n^3)$.
- This paper introduces two novel techniques: *Lazy Cycle Detection* and *Hybrid Cycle Detection* which significantly speed up the detection of cycles in a constraint graph.
- The memory consumption of various analyses have been studied, while exploring new data structures to represent points-to sets.

Related Work

- Andersen's analysis involved finding an iterative fixed point solution to a set of equations imposed by constraint vectors.
- Most algorithms revolve around novel optimisations to compute the dynamic transitive closure of the constraint graph.
  1. *Faehndrich et al*: Performs an artificially restricted DFS, resulting in incompleteness.
  2. *Heintze and Tardieu* (HT) : Indirect constraints are not added to the graph,and resolved by reachability queries. Works fine for field based implementation.
  3. *Pearce et al.* (PKH): Two algorithms: one that improves 1 by maintaining a topological ordering. One that periodically sweeps and checks for cycles.
  4. *Berndl et al.* (BLQ): Uses a field insensitive inclusion based analysis with BDDs for Java. Does not handle indirect calls.
  5. *Das*: A one level flow analysis which disregards multi-level pointers. Similar to Steensgaard's analysis but fails for Java and C++.

Proposed Solutions

- The most natural algorithm for Dynamic transitive closure uses a simple work list. It runs until convergence in $O(n^3)$.
- **Why** do we attempt to detect cycles?
  1. All nodes in a cycle have the same points-to set because of the transitive property.
  2. Such nodes can be collapsed into one node to reduce the graph size.
  3. A fine balance should be drawn between searching for cycles too early and too late.
- *Lazy Cycle Detection*(LCD) attempts to detect cycles only when they are likely to be found. The underlying assumption is as stated above.
- The algorithm ensures that it never processes the same edge twice. Thus, a cycle is deemed to be detected only when the points-to sets are equal and the edge is not processed.
- Cycles may possibly be detected later than usual.

Proposed Solutions: Continued

- Online cycle detection requires multiple graph traversals, making it costly.
- *Hybrid Cycle Detection*(HCD) builds an offline constraint graph by adding a suitable set of nodes and edges for constraint relations.
- Strongly Connected Components are computed using *Nutiila et al.*'s augmentation of Tarjan's algorithm by using union find.
- Reference nodes added may pose a problem, as their information is futuristic, meaning that they are deterministic only upon termination.
- By augmenting the algorithm to consider the current points-to set of each reference node, this issue can be mitigated.
- HCD does not find all cycles, but detects the ones it does find earliest and with no online graph traversal.

Evaluation and Results

- The source code uses techniques from existing papers to handle indirect calls and speeding up the work list.
- The tests were run on a dual core 1.83 GHz processor having 2GB memory. gcc-4.1.1 with O3 and Ubuntu 6.10 were used.
- It is observed that when run on standard benchmark programs, HT beats all existing algorithms.
- LCD is 1.05x faster and uses 1.2x less memory than HT. HCD is 1.8x slower and uses 1.4x more memory ,but is 1.9x faster than PKH.
- Coming to memory usage, BDDs are 2x slower but use 5.5x less memory than bitmaps.
- Three principle factors govern relative performance:
  1. Number of collapsed nodes as a part of SCCs.
  2. Number of nodes searched in Depth First Order
  3. The extent of propagation of points-to information across the graph.

Summary and Future Work

- This paper presents an efficient way to scale up points-to analysis.
- Two novel approaches namely LCD and HCD have been discussed.
- These approaches seek to ease cycle detection and enhance equivalence of nodes in the constraint graph.
- Programs having almost 2.7M lines of code have been tested.
- Obtaining a deeper understanding of memory usage by HCD to enhance performance can be one area to explore.
- Making the analysis partially context sensitive by introducing a topological ordering of method processing can be taken up.

Thank You!