Elimination-based Range Analysis for Unstructured code in the LLVM framework

Paul Subotic

November 17, 2011

Introduction

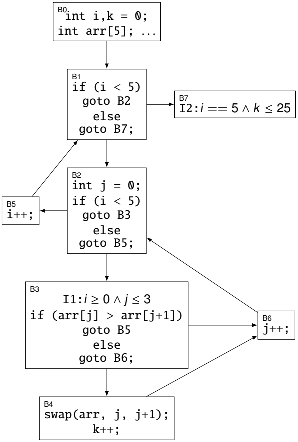
- Range Analysis
  - Finds *lower* and *upper* bounds of variables values
- Challenges
  - Conceptionally infinitely ascending chains
  - Identify Loops
- Existing techniques
  - Relies on code structure (e.g. Astrée [Cousot et al., 2006])
  - Require a pre-processing stage to discover loop headers ([Bourdoncle, 1993])

Introduction

- Our technique:
  1. Extends elimination-based data flow analysis to a lattice with infinite ascending chains
  2. Fast termination
  3. Loops are detected intrinsically with in the data flow analysis.

- Implemented as an analysis pass in the LLVM compiler framework.

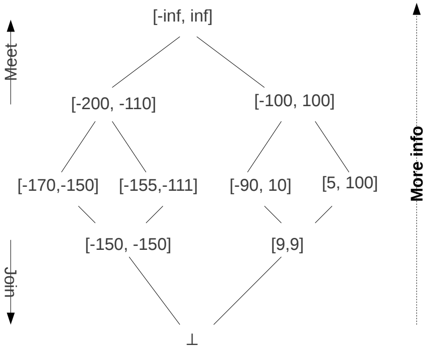
Motivating Example



Foundations

- Range Analysis is a complete lattice
- $x \sqsupseteq y$, x is as or less precise than y
- $\top$ least element (least precise),
- $\bot$ greatest element, so $\top \sqsupseteq \bot$
- $\sqcup$ merges information
- $\sqcap$ constrains information

Representing Information with Intervals



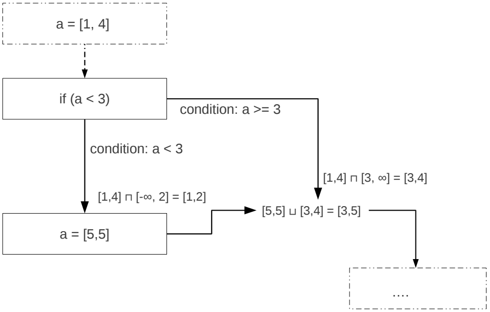
Some Existing Techniques

- **Iterative Data-Flow Analysis [Kildall, 1973]** :
  - A technique for iteratively gathering variable information at various points in a computer program.
  - Operates on finite and short lattice structures
- **Abstract Interpretation [Cousot & Cousot, 1977]** :
  - A theory of sound approximation of the semantics of computer programs
  - Approximating the execution behaviour of a computer program
  - Additional theory of widening/narrowing to accelerate convergence, required with high and unbounded domains

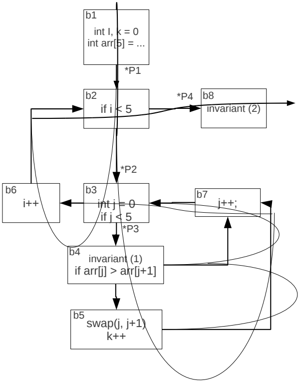
Iterative Data-Flow Analysis

- Input in the form of a Control Flow Graph (CFG)
- Initialise to $\perp$
- Every block transforms the values
- Iterate through CFG until a fixpoint is reached

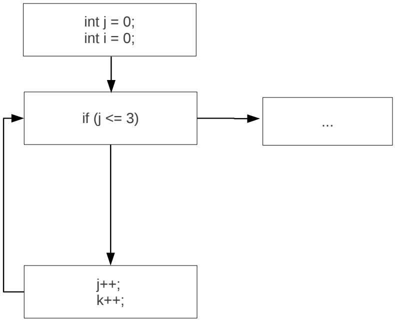
Attempt 1: Iterative Data-Flow Analysis



Attempt 1: Iterative Data-Flow Analysis



With Kleene Iteration



With Kleene Iteration

$\forall l_i \in \mathcal{L}. l_1 \sqsubseteq l_2 \sqsubseteq l_3 \sqsubseteq l_4 ... \sqsubseteq l_n$
where:
In the example, when the inner loop is first visited, we have that $j \mapsto [0, 0]$ and $k \mapsto [0, 0]$. In subsequent visits,

$$\begin{gathered} j \mapsto [0,1] \text{ and } k \mapsto [0,1], \\ j \mapsto [0,2] \text{ and } k \mapsto [0,2], \\ j \mapsto [0,3] \text{ and } k \mapsto [0,3], \\ \vdots \\ j \mapsto [0,4] \text{ and } k \mapsto [0,\infty]. \end{gathered}$$

The Problem: Slow Termination

- Impractically slow termination
  - Conditions not incorporating increasing variables
  - Large loop bounds

Attempt 2: Abstract Interpretation

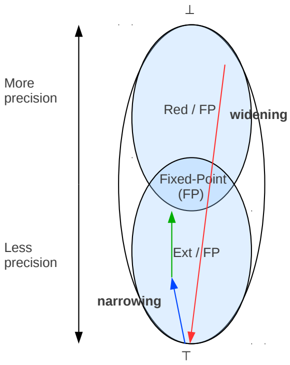
- General method to compute a sound approximation of program semantics
  - Define an abstract semantics, soundly connect to the concrete semantics
  - Soundness ensures that if a property does not hold in the abstract world, it will not hold in the concrete world
  - Define widening and narrowing operator

Abstract Interpretation

Widening and narrowing enforce termination

- Widening safely approximates the fixpoint solution
- Narrowing recovers some precision

Attempt 2: Abstract Interpretation



Abstract Interpretation

- Requires to know where to perform widening
- Previously approaches
  - Use the syntax to determine the loop
  - Perform complicated pre-processing to find loop headers

Our Approach

- Discovers loops implicitly using elimination-based data flow analysis
- Various acceleration techniques can be embedded such as widening and narrowing

Our Approach

- Elimination-based approach: Based on Gaussian elimination
- Instead of iterating, we eliminate variables from the flow equations
  - substitution
    e.g. $x = \texttt{true}, y = x \vee \texttt{false} \rightsquigarrow y = \texttt{true} \vee \texttt{false}$
  - loop-breaking
    e.g. $x = x \wedge \texttt{true} \rightsquigarrow x = \texttt{true}$
- When all variables are eliminated, we compute a solution

Elimination-based Approach Example - Diverging

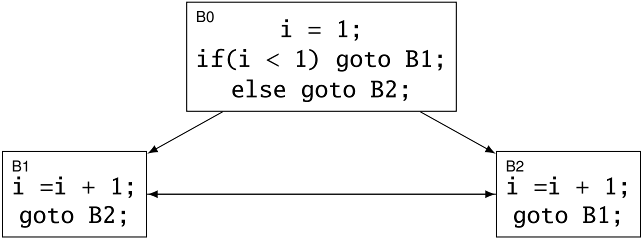


Figure: An Irreducible CFG of a Diverging Program

Elimination

$$
\begin{aligned}
\text{EQS} &= \begin{cases} X_0 = f_0(\top) \\ X_1 = f_1(X_0, X_2) \\ X_2 = f_2(X_0, X_1) \end{cases} \\
\text{Substitution} \rightsquigarrow & \\
\text{EQS}_0 &= \begin{cases} X_0 = f_0(\top) \\ X_1 = f_1(f_0(\top), X_2) \\ X_2 = f_2(f_0(\top), X_1) \end{cases} \\
\text{Substitution} \rightsquigarrow & \\
\text{EQS}_1 &= \begin{cases} X_0 = f_0(\top) \\ X_1 = f_1(f_0(\top), X_2) \\ X_2 = f_2(f_0(\top), f_1(f_0(\top), X_2)) \end{cases} \\
\text{Break Loop, Substitute Back} \rightsquigarrow & \\
\text{EQS}_2 &= \begin{cases} X_0 = f_0(\top) \\ X_1 = f_1(f_0(\top), F^*(f_2(f_0(\top), f_1(f_0(\top), X_2), X_2'))) \\ X_2 = F^*(f_2(f_0(\top), f_1(f_0(\top), X_2), X_2')) \end{cases}
\end{aligned}
$$

Solve

- $X_1 = f_1(f_0(\top), F^*(f_2(f_0(\top), f_1(f_0(\top), X_2), X_2')))$
- F^* performs widening and narrowing

An Example

LLVM Prototype

- Implemented in LLVM for core instructions
- Implementation supports both Intervals and Symbolic Intervals

Block	i	j	k
B0	[0, 0]	$\perp$	[0,0]
B1	[0, 5]	[0, 5]	[0, ∞]
B2	[0, 4]	[0, 0]	[0, ∞]
B3	[0, 4]	[0, 5]	[0, ∞]
B4	[0, 4]	[1, 4]	[1, ∞]
B5	[1, 5]	[5, 5]	[1, ∞]
B6	[5, 5]	[5, 5]	[0, ∞]

Table: Motivating Example

Test	Exact	Bounded	Part Widen	Full Widen
T1	1	5	0	0
T2	2	1	0	0
T3	2	1	0	0
T4	1	3	2	0
T5	0	10	0	0
T6	3	1	0	0
T7	1	2	0	0
T8	4	4	5	0
T9	1	0	0	5
T10	1	0	4	0
T11	2	2	0	0
T12	2	3	3	1
T13	1	2	2	0
T14	3	6	6	0
T15	3	5	4	0
All	27	45	26	6
(%)	26	43	25	6

Table: Variable Bounds Per Test Case

Summary

- Implemented in the LLVM Compiler Framework
- Feasibility shown using several test programs

Some Future Work

- Conduct comparison with existing techniques
- Add non-numerical domains
- Improve precision through additional abstract domains (Template Polyhedra [Sankaranarayanan et al., 2005])
- Integrate with acceleration methods such as policy iteration [Gawlitza & Seidl, 2007]

References I

Bourdoncle, F. (1993).
Efficient chaotic iteration strategies with widenings.
In *In Proceedings of the International Conference on Formal Methods in Programming and their Applications* (pp. 128–141).: Springer-Verlag.

Cousot, P. & Cousot, R. (1977).
Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints.
In *4th POPL* (pp. 238–252).

References II

Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., & Rival, X. (2006).
Combination of abstractions in the ASTREÉ static analyzer.
In *11th ASIAN* (pp. 272–300).

Gawlitza, T. & Seidl, H. (2007).
Precise fixpoint computation through strategy iteration.
In *Proceedings of the 16th European conference on Programming*, ESOP'07 (pp. 300–315). Berlin, Heidelberg: Springer-Verlag.

Kildall, G. A. (1973).
A unified approach to global program optimization.
1st POPL (pp. 194–206).

References III

Sankaranarayanan, S., Sipma, H. B., & Manna, Z. (2005).
Scalable analysis of linear systems using mathematical programming.
In *In Proc. VMCAI, LNCS 3385* (pp. 25–41).: Springer.