Automatic Generation of Bit-vector Analysis Using OpenAnalysis

Shweta Behere, Dr. Michelle Strout, Dr. Paul Hovland

- **Why do we need program analysis ?**

Effective program analysis is essential for program optimization and program understanding. It is needed for software engineering tasks such as program verification and error detection.

- **What is bit-vector analysis ?**

Bit-vector analysis is an analysis that uses bits to store information at each program point. Dataflow information can be encoded with sets having true or false values with a bit-vector. Examples of bit-vector analysis are reaching definitions, liveness and available expressions.

- **What are problems with classic bit-vector analyses ?**

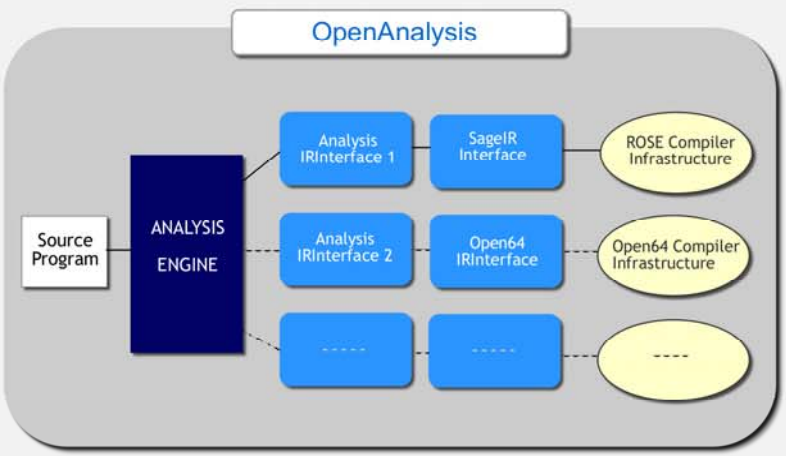
Classic bit-vector analyses techniques assume only scalar variables within a program. However, real programs consist of pointer usage, structures and arrays.

- **Why do we need automatic generation ?**

Bit-vector analyses are generally hand-written for a particular language. If we are able to generate a bit-vector analysis by providing a specification, it would be of great help to software engineers.

- **What is OpenAnalysis(OA) ?**

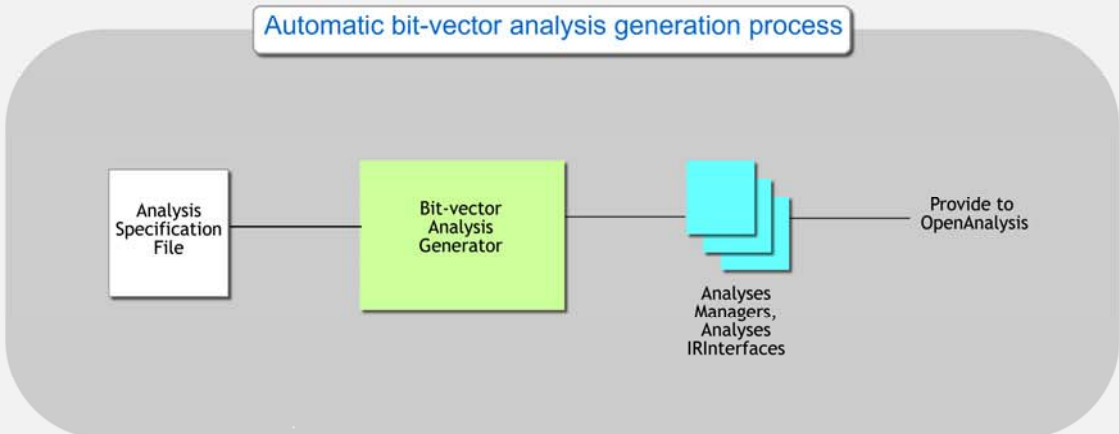
OA is an open source analysis toolkit that separates program analysis from program representation. OA can be coupled with any imperative or imperative-based object-oriented programming language, by implementing various abstract interfaces.



Approach:

One approach is to lower the higher-level semantics involving structures and arrays to a representation that has only scalar temporaries and accesses to memory. A conservative analysis assumption would be that all accesses to memory possibly overlap. Language-specific transformations must be performed on higher-level representations; therefore the lowering approach is not always applicable.

Our aim is to effectively analyze programs containing aliases, arrays and other complex structures at a higher semantic level, while still using data-flow analysis specifications for scalars. In our approach, we describe the data-flow analysis specification in a set-based specification language, and automatically generate an analysis implementation that uses aliasing information to obtain precise dataflow analysis results.



Specification for Liveness:

meet: union
direction: backward
type: may
gen[s]: {uses[s]}
kill[s]: {defs[s]}

Example for Liveness:

a = 5;
*p = 4;

As Liveness is a may analysis, a is still live after the last statement.

Solution:

Use the upper bound or lower bound based on the may/must information.

Liveness is a may analysis. We will take the upper bound for the gen set (*p could be everything) and the lower bound for the kill set (*p could be nothing).

Future Work:

- Provide options for the user to generate context sensitive/insensitive analysis
- Extend implementation generation technique to any domain specific data-flow analysis

Related Work:

A generalized theory of bit vector data flow analysis - Uday P. Khedker, Dhananjay M. Dhamdhere

They present a theory for bit-vector data flow analysis and characteristics of bit-vector frameworks.