Compositional Separate Modular Static Analysis of Programs

Patrick COUSOT
École Normale Supérieure
45 rue d'Ulm
75230 Paris cedex 05, France
mailto:Patrick.Cousot@ens.fr
http://www.di.ens.fr/~cousot

Radhia COUSOT
École Polytechnique
91128 Palaiseau cedex, France
mailto:Radhia.Cousot@polytechnique.fr
http://lix.polytechnique.fr/~rcousot

SSGRR'2001, L'Aquila, Italy August 6–12, 2001

Introductive Motivations

Program Static Analysis

- Static program analysis is the automatic compile-time determination of run-time properties of programs;
- Used in many applications from optimizing compilers, to abstract debuggers and semantics based program manipulation tools (such as partial evaluators, error detection and program understanding tools).

Abstract Interpretation

- Supporting theory;
- General idea: a program static analyzer computes an effective approximation of the program semantics (semantics = formal specification of all possible run-time behaviors).

Principle of Program Static Analysis

In order to determine runtime properties of a program P, a static analyzer:

- inputs the program P;
- builts a system of equations/constraints $X \sqsupseteq F[\![P]\!]X$;
- solves it $A \sqsupseteq \mathit{lfp}\, F$;
- outputs the solution A (in some user understandable form).

Example: Interval Analysis [1]

program	equations	solution
x := 1;		
1:		
while x < 10000 do	$X_1 = [1, 1]$	$A_1 = [1, 1]$
2:	$X_2 = (X_1 \cup X_3) \cap [-\infty, 9999]$	$A_2 = [1, 9999]$
x := x + 1	$X_3 = X_2 \oplus [1, 1]$	$A_3 = [2, 10000]$
3:	$X_4 = (X_1 \cup X_3) \cap [10000, +\infty]$	$A_4 = [10000, 10000]$
od;		
4:		

[1] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Global analysis

Principle of Global Analysis

- A global system of equations/constraints is established for the whole program;
- This system of equations is solved iteratively at once (using various chaotic iteration strategies).

Advantages/Drawbacks of Global Analysis

- Simple and can be made very precise;

but
- The program hence the system of equations can be very large;
- The convergence of the iterates may be slow;
- The whole program must be reanalyzed even if a small part only is changed;

so
- Either less precise global analyzes;
- Or better, separate modular local analyses;

The Problem

The Problem Considered in the Paper

Design methods for compositional separate modular static analysis of programs.

Separate Local Analysis

Principle of (Ideal) Separate Analysis

- The program $P[P_1, \ldots, P_n]$ is decomposed into parts P_1, ..., P_n (such as functions, procedures, modules, classes, components, libraries, etc.);
- The parts are analyzed separately: $A_i \sqsupseteq \mathit{lfp}^{\sqsubseteq_i} F[\![P_i]\!]$, $i = 1, \ldots, n$
- The whole program is analyzed by composing the analyzes of the parts: $A \sqsupseteq \mathit{lfp}^{\sqsubseteq} F[\![P]\!][A_1, \ldots, A_n]$.

Advantages of Separate Analysis

- Memory saving: the whole-system of equations/constraints does not need to fully reside in memory at the same time;
- Time saving: The separate analyses of the parts can be done in parallel;

but

- In general the analyzes of the parts are interdependent:

  $A_i \sqsupseteq \mathit{lfp}^{\sqsubseteq_i} \boldsymbol{\lambda} X_i \cdot F[\![P_i]\!]\langle Y, X_1, \ldots, X_i, \ldots, X_n \rangle$

  Y: dependence on the global program elements;

  $X_k, k = 1, \ldots, i-1, i+1, \ldots, n$: dependence of part P_i on the other program parts.

Proposed Separate Analysis Methods

A global whole-program analysis can be decomposed into separate analyses, by one of the following methods:

- Simplification-based separate analyses;
- Worst-case separate analyses;
- Separate analyses with (user-provided) interfaces;
- Symbolic relational separate analyses;
- Composition of the above separate local analyses and global analysis methods.

Simplification-Based Separate Analysis

Principle of Simplification-Based Separate Analysis

- When handling a program part P_i, just simplify the equations/constraints into $X \sqsupseteq_i F_s[\![P_i]\!](X)$;
- Wait for the whole-program before computing the solution for parts together with the global solution:
  $$A \sqsupseteq \mathit{lfp}^{\sqsubseteq} F_s[\![P]\!][\mathit{lfp}^{\sqsubseteq_1} F_s[\![P_1]\!], \ldots, \mathit{lfp}^{\sqsubseteq_n} F_s[\![P_n]\!]];$$

variant:
- Use a preliminary simpler whole-program analysis to help the simplification process.

Advantages/Drawbacks of Simplification-Based Separate Analysis

- The simplification is cheap and improves the later iterative fixpoint computation cost;

but:

- Negligible benefit when compared to the cost of the iterative fixpoint computations;
- Does not scale up for very large programs;

Worst-Case Separate Analysis

Principle of Worst-Case Separate Analysis

- Assume absolutely no information is known on the global program elements and on the other program parts:

$$A_i \sqsupseteq \mathit{lfp}^{\sqsubseteq_i} \boldsymbol{\lambda} X_i \cdot F[\![P_i]\!]\langle \top, \top, \ldots, X_i, \ldots, \top \rangle$$

($\top$ denotes the absence of information).

Advantages/Drawbacks of Worst-Case Separate Analysis

- Very efficient (the analyzes of the parts can be done in parallel before the global analysis of the main program);

but

- Quite imprecise.

Separate Analysis with (User-Provided) Interfaces

Principle of Separate Analysis with (User-Provided) Interfaces

- Ask the user which *assumptions* can be made on other parts $P_1, \ldots, P_{i-1}, P_{i+1}, \ldots, P_n$ when analyzing part P_i;
- Check that the analysis of part P_i *guarantees* that the assumptions made by the other parts on P_i are satisfied;
- Otherwise ask the user to provide more precise information on the interfaces between the program parts;

variant: • Generate (part of) the interfaces automatically (e.g. types).

Advantages/Drawbacks of Separate Analysis with (User-Provided) Interfaces

- Can always be made as precise as a global analysis;
- Much more efficient;

but:
- A large burden on the user.

Symbolic relational separate analysis

Principle of Symbolic Relational Separate Analysis

- Name the external objects and operations used by a program part;
- Relate them to internal objects by analysis of the internal operation done on external objects;
- Delay the analysis of the external effects as much as possible.

Example of Symbolic Relational Analysis

```
procedure Hanoi (n : integer; var a, b, c : integer; var Ta, Tb, Tc : Tower);
begin
  { n = n0 ∧ a = a0 ∧ b = b0 ∧ c = c0 }
  if n = 1 then begin
        b := b + 1; Tb[b] := Ta[a]; Ta[a] := 0;a := a − 1;
      { n = n0 = 1 ∧ a = a0 − 1 ∧ b = b0 + 1 ∧ c = c0 }
  end else begin
      Hanoi(n − 1, a, c, b, Ta, Tc, Tb);
      { n = n0 > 1 ∧ a = a0 − n + 1 ∧ b = b0 ∧ c = c0 + n − 1 }
      b := b + 1; Tb[b] := Ta[a]; Ta[a] := 0;a := a − 1;
      { n = n0 > 1 ∧ a = a0 − n ∧ b = b0 + 1 ∧ c = c0 + n − 1 }
      Hanoi(n − 1, c, b, a, Tc, Tb, Ta);
       { n = n0 > 1 ∧ a = a0 − n ∧ b = b0 + n ∧ c = c0 }
  end;
  { n = n0 ≥ 1 ∧ a = a0 − n0 ∧ b = b0 + n0 ∧ c = c0 }
end;
```

Example of Symbolic Relational Analysis, Con'd

```
a := n; b := 0; c := 0;
```
$\{\ n = a \ \wedge\ b = 0 \ \wedge\ c = 0\ \}$

```
Hanoi(n, a, b, c, Ta, Tb, Tc);
```
$\{\ \exists n_0, a_0, b_0, c_0 : n_0 = a_0 \ \wedge\ b_0 = 0 \ \wedge\ c_0 = 0 \ \wedge$

$n = n_0 \geq 1 \ \wedge\ a = a_0 - n_0 \ \wedge\ b = b_0 + n_0 \ \wedge\ c = c_0\ \}$

This last post-condition can be simplified by projection as:

$\{\ a = 0 \ \wedge\ n = b \geq 1 \ \wedge\ c = 0\ \}$

Advantages/Drawbacks of Symbolic Relational Analysis

- Fully automatic (no human interaction);
- Very powerful;

but:
- Relational analyzes can be very expensive;
- If nothing is known about the other program parts everything may end up being delayed until the global analysis (e.g. virtual methods in object-oriented languages).

Composition of Separate Local and Global Analyses

Principle of Separate Local and Global Analysis Composition

In practice, a good combination of the previous methods is necessary. For example:

- Create parts through cutpoints;
- Preliminary global analysis and simplification;
- Refine the abstract domain into a symbolic relational domain;
- Iterated separate program static analysis starting from worst-case;

Example: Iterated Separate Program Static Analysis

- Start with a worst case assumption $Y^0 = \top$, $X_1^0 = \top$, ..., $X_n^0 = \top$ (or user-provided assumptions);
- Iterate a separate analysis with interfaces:

$$X_i^{k+1} = \mathit{lfp}^{\sqsubseteq_i} \boldsymbol{\lambda} X_i \cdot F[\![P_i]\!]\langle Y^k, X_1^k, \ldots, X_i, \ldots, X_n^k\rangle \qquad i = 1, \ldots, n$$

$$Y^{k+1} = \mathit{lfp}^{\sqsubseteq} \boldsymbol{\lambda} Y \cdot F[\![P[P_1, \ldots, P_n]]\!]\langle Y, X_1^k, \ldots, X_n^k\rangle$$

Advantages/Drawbacks of Iterated Separate Program Static Analysis

- The iteration can be expansive;

but:
- The iteration can be stopped at any step (e.g. when getting out of time);

Conclusion

- Many variants are presented in the paper (together with references);
- Presently one can globally analyze a few 100 000 lines of code in few minutes to hours;
- Already effective methods so it's time to think to Internet applications;
- More work and experimentation on separate analysis is needed to deal with a few 1 000 000 lines;

THE END, THANK YOU.