Exploiting Abstract Interpretation for Model Checking Programs

Gilbert Pajela, Jordi Navarrette, and Subash Shankar
City University of New York (CUNY)

February 25, 2019

Objective I

- Objective: to improve the performance of software model checking using static analysis techniques
- Static analysis and model checking: two formal verification techniques that can be used to verify that a program adheres to its specification
- Our new method of combining static analysis with model checking:
  - Has the potential to improve the performance of tools that use formal verification techniques
  - Is automatable, and we are currently working on its automation

Objective II

- Basic outline of steps in our method:
  - Perform a static analysis of a C program
  - Insert the results of the analysis into the program
  - Run a model checking tool with the modified program as input
- Test results: significant improvement in speed of the model checker using our method on some programs

Our Approach

1. Performing an abstract interpretation to identify variable values at varied program points
2. Using backward slicing to choose which program variables to track
3. Exploiting this information to reduce the search space of the model checker

Approaches to Program Verification

- **Abstract Interpretation**: An approximation of program semantics based on mappings between concrete and abstract lattices $\Rightarrow$ symbolic evaluation in abstract domain
  - ☹ Usefulness of [nondeterministic, lossy] abstract program dependent on abstractions
  - ☹ Loops may require unrolling, with loss of precision (or an indeterminate fixed point computation)
- **Model Checking with CEGAR**: Iteration over abstraction - model checking - refinement cycle to automatically prove program correctness
  - ☹ State space explosion
  - ☹ Success limited by choice of predicate abstractions

Existing Tools

- Frama-C [2]:
  - An extensible C verification framework
  - Plugins include abstract interpretation (Eva) and slicing
- CPAchecker [1]:
  - Configurable program analysis dealing mainly with model checking of control-flow automata constructed from C programs
  - Includes support for CEGARish checking (in `predicateAnalysis` configuration)
- CegarMC [3]:
  - A previously published Frama-C plugin by the authors
  - Integrates CEGAR-based model checkers into Frama-C

Static Analysis Tools I

- Two Frama-C plugins used by our method:
  1. Eva to automatically compute sets of possible values for the variables of an analyzed program
  2. The program slicing plugin:
     - Reduces a program based on a backward slicing criterion, traditionally a program location and a set of program variables, so that the behavior of the original program is preserved with respect to the criterion
     - The results from Eva are also used to compute program slices

Static Analysis Tools II

- Sets of possible values introduced into the program using assume statements
- Assume statements inserted at selected points throughout the program
- CPAchecker run with the modified program as input

Example

```
int x, x0, y, y0;
y = 0;
while (x > 0) {
  x0 = x;
  y0 = y;
  x = x - 1;
  y = y + 2;
  if (2 * (x0 - x) != y - y0)
    error();
}
```

Example With Assume Statements Inserted I

```
int x, x0, y, y0;
y = 0;
assume(x >= INT_MIN && x <= INT_MAX &&
       x0 >= INT_MIN && x0 <= INT_MAX &&
       y == 0 && y0 == 0);
while (x > 0) {
  assume(x >= 1 && x <= INT_MAX &&
         x0 >= INT_MIN && x0 <= INT_MAX &&
         y >= 0 && y <= INT_MAX - 1 &&
         y % 2 == 0 &&
         y0 >= 0 && y0 <= INT_MAX - 1 &&
         y0 % 2 == 0);
  x0 = x;
  y0 = y;
```

Example With Assume Statements Inserted II

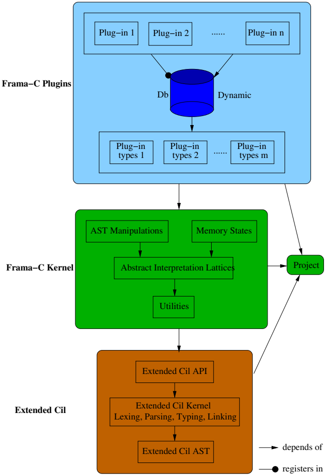
```
x = x - 1;
y = y + 2;
if (2 * (x0 - x) != y - y0)
  error();
assume(x >= 0 && x <= INT_MAX - 1 &&
       x0 >= 1 && x0 <= INT_MAX &&
       y >= 2 && y <= INT_MAX - 1 &&
       y % 2 == 0 &&
       y0 >= 0 && y0 <= INT_MAX - 1 &&
       y0 % 2 == 0);
```

Example With Assume Statements Inserted III

```
}
assume(x >= INT_MIN && x <= 0 &&
       x0 >= INT_MIN && x0 <= INT_MAX &&
       y >= 0 && y <= INT_MAX - 1 &&
       y % 2 == 0 &&
       y0 >= 0 && y0 <= INT_MAX - 1 &&
       y0 % 2 == 0);
```

TOOL DEMO

Frama-C Architecture I

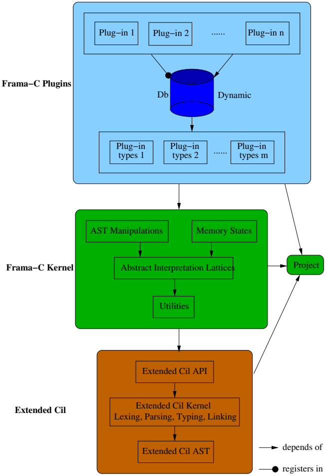


From Frama-C Plugin Manual

Plugins:

- Interfaces to abstract syntax tree (AST), C intermediate language (CIL), AI lattices, etc. provided by kernel
- Plugins used for either analysis (≥ 1 AST) or source-to-source transformation (> 1 AST)
- Kernel-integrated plugins include Eva and wp (statically linked)

Frama-C Architecture II

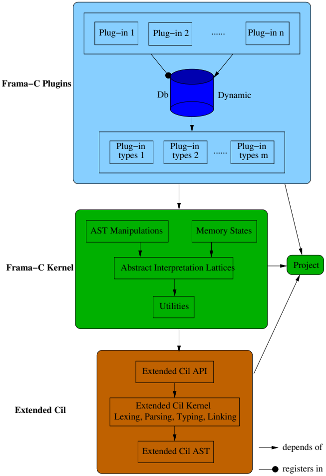


From Frama-C Plugin Manual

Plugins:

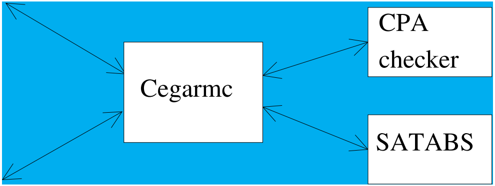
- Extensible through user-written plugins, typically linked dynamically
- Common plugin interface allows for inter-plugin information sharing, along with a central mechanism for combining results
- All programmed in OCAML

Tool Architecture



From Frama-C Plugin Manual

CegarMC Plugin



Ongoing Research I

- Fine-tune the abstract interpretation
  - E.g., the program points where the abstract interpretation information is exploited
- Evaluate our method with more and a wider variety of example programs
- Fully automate our method, extending our CegarMC plugin

Ongoing Research II

- Explore other possible combinations of abstract interpretation and model checking:
  - **Residual program**: unexplored parts of a model check
  - Create a residual program generator using Frama-C plugins
  - Pass the residual program generated by CPAchecker along with any other necessary information to a Frama-C plugin

References

[1] D. Beyer and M. E. Keremoglu.
CPAchecker: A tool for configurable software verification.
In *Computer Aided Verification (CAV)*, pages 184–190, 2011.

[2] F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski.
Frama-C, a software analysis perspective.
In *Formal Aspects of Computing*, 27: pages 573–609, March 2015.

[3] S. Shankar and G. Pajela.
A Tool Integrating Model Checking into a C Verification Toolset.
In *International Symposium on Model Checking Software*, pages 214–224, 2016.

Thank you!

Questions?

Frama-C: downloadable from `www.frama-c.com`
CPAchecker: downloadable from `cpachecker.sosy-lab.org`
CegarMC Plugin: downloadable from
`http://www.compsci.hunter.cuny.edu/~sshankar/cegarmc.html`