ESBMC 5.0

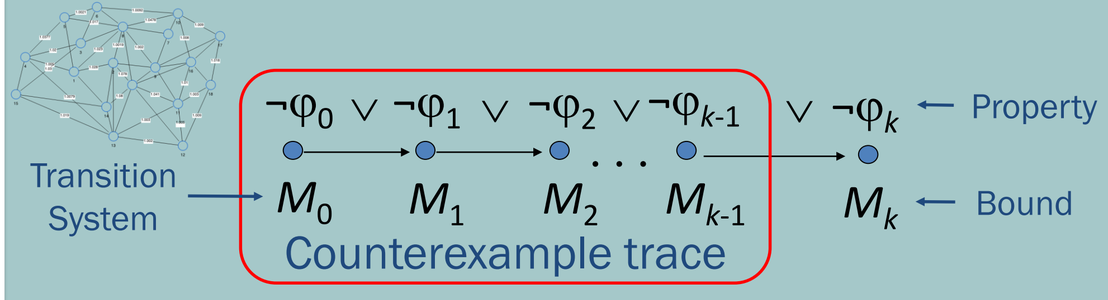
An Industrial-Strength C Model Checker

Mikhail R. Gadelha, **Felipe R. Monteiro**, Jeremy Morse,
Lucas Cordeiro, Bernd Fischer, Denis A. Nicole
University of Southampton, Federal University of Amazonas, University of Bristol,
University of Manchester, University of Stellenbosch

I. Motivation

"... proving a software system correct requires much more effort, knowledge, training, and ingenuity than writing the software in trial-and-error style."

- E. M. Clarke *et al.*, Handbook of Model Checking 2018.



Bounded model checking

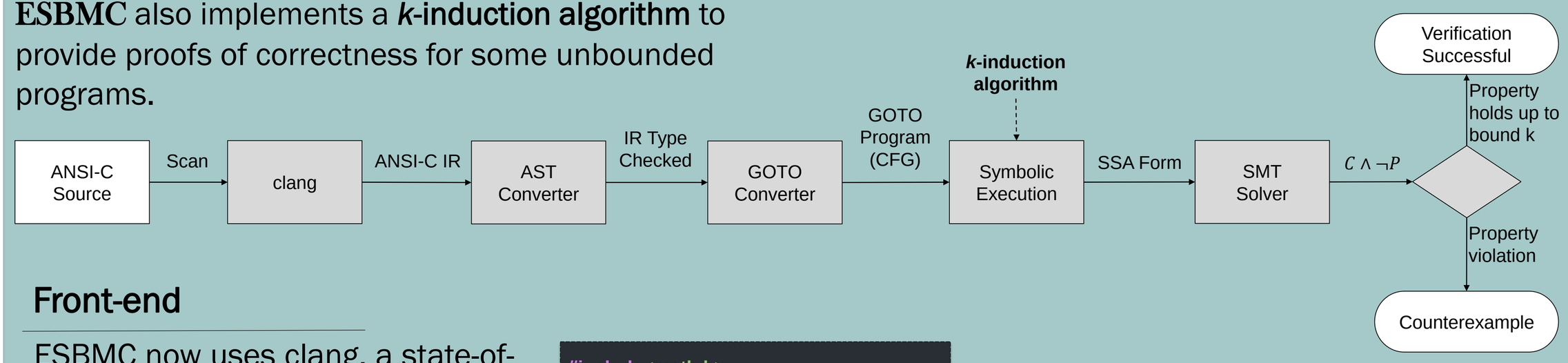
Translates the model into a VC ψ such that: ***ψ* is satisfiable iff *φ* has counterexample of max. depth *k***

ESBMC is an open source, permissively licensed (apache 2), bounded model checker (BMC) for **C programs.** It is written primarily in portable C++ and, using Autotools, builds on multiple platforms.

The tool was developed for bounded model checking of both **sequential and concurrent programs** using a variety of **SMT solvers**, and has a proven track record of bug finding in real world applications.

II. Components & Features

ESBMC also implements a ***k*-induction algorithm** to provide proofs of correctness for some unbounded programs.



Front-end

ESBMC now uses clang, a state-of-the-art compiler suite for C/C++/ObjectiveC/ObjectiveC++ widely used in industry, as its front-end.

Control-Flow Graph Generator

It takes the program AST and transforms it into an equivalent GOTO program: a simplified representation that consists only of assignments, conditional and unconditional branches, assumes, and assertions.

Symbolic Execution Engine

ESBMC symbolically executes the GOTO program.

It unrolls loops *k* times, generates the SSA form of the unrolled program, and derives all the safety properties to be checked by the SMT solver.

SMT Back-end

ESBMC's SMT back-end supports five solvers: Boolector (default), Z3, MathSAT, CVC4 and Yices.

Python API

ESBMC now includes a **Python API** that reduces the difficulty of prototyping new features and makes the tool internals accessible to a wider audience.

k-Induction

$$kind(P,k) = \begin{cases} P\ contains\ a\ bug, & if\ B(k)\ is\ SAT \\ P\ is\ correct, & if\ B(k) \wedge [F(k) \vee I(k)\ ] is\ UNSAT \\ kind(P,k+1), & otherwise \end{cases}$$

Floating-point Encoding

ESBMC encodes floating-point arithmetic using:

- **bitvectors**, which extends the floating-point arithmetic support to all solvers that are currently integrated.
- the **SMT theory of floating-points**, available only in Z3 and MathSAT.

```
#include<math.h>
#include<assert.h>
int main() {
  unsigned int N = nondet_uint();
  double x = nondet_double();
  if(x <= 0 || isnan(x))
    return 0;
  unsigned int i = 0;
  /*i = nondet_uint();*/
  /*x = nondet_double();*/
  /*__ESBMC_assume(i < N);*/
  while(i < N) {
    x = (2*x);
    assert(x>0);
    ++i;
  }
  /*__ESBMC_assume(!(i < N));*/
  assert(x>0);
  return 0;
}
```

III. SV-COMP 2018

The ESBMC's *k*-induction version achieved a score of 5476 and third place overall.

The k-induction algorithm reported 4301 correct results, with 92% of witnesses being correctly validated.

None of the wrong results were related to the k-induction algorithm.

IV. Future Work

We are extending the *k*-induction algorithm to reuse information from the inductive step, to make bug finding more efficient.

For further information, publications, and downloads, see:
http://www.esbmc.org/ https://github.com/esbmc/esbmc