Handling Loops in Bounded Model Checking using *k*-Induction and Interval Analysis

UNIVERSITY OF Southampton

Mikhail Y. R. Gadelha. Supervisors: Denis A. Nicole, Genarro Parlato

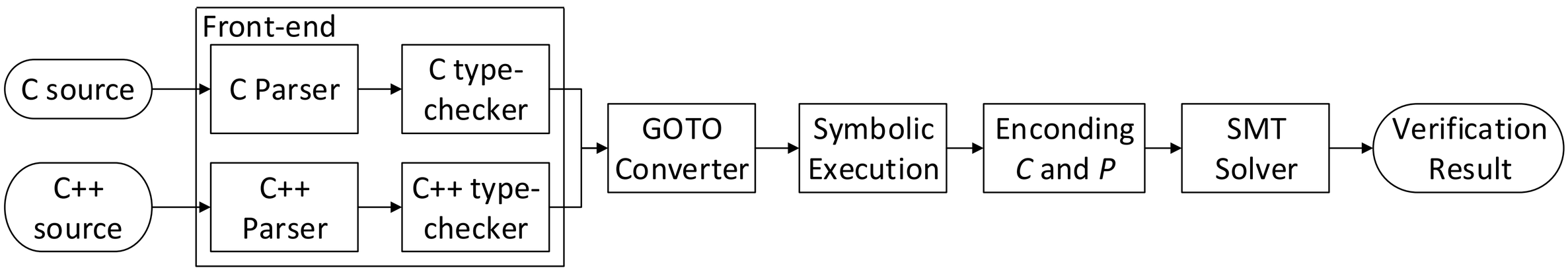
Motivation

Bounded Model Checking (BMC) techniques based on Boolean Satisfiability (SAT) or Satisfiability Modulo Theories (SMT) have been successfully applied to verify single and multithreaded programs and to find subtle bugs in real programs. The idea of the BMC techniques is to check the negation of a given property at a given depth, i.e., to find bugs in a program up to a limit of iterations *k*.

Typically, the BMC techniques are only able to falsify properties up to a given depth k; they are not able to prove the correctness of the system, unless an upper bound of k is known, i.e., a bound that unfolds all loops and recursive functions to their maximum iteration.

We are looking at interval and complex hull analysis linked with k-induction (loop unrolling) to try and automate loop verification by induction. Interval analysis can automatically imply termination conditions; k-induction can simplify loop invariants to the extent that they can be guessed syntactically.

The ESBMC Model Checker



Fibonacci Example

```
integer fib(integer n) { return n < 3 ? 1 : fib(n-2)+fib(n-1); }

integer myfib(integer n) {
  integer c=1; p=1; i=2;
  while(i<n) {
    integer t=p+c;
    p=c;
    c=t;
    i++;
  }
  assert (c == fib(n)); // Post condition
  return c;
}
```

Base Case for k <= 3

```
integer myfib(integer n) {
  assume( n <= 3 );
  integer c = 1, p = 1, i = 2;
  if( i < n ) {
    integer t = p + c;
    p = c;
    c = t;
    i++;
    assume( i < n + 1 );
    assert( c == fib(i) );
  }
}
```

The invariant (bold) is generated by interval analysis, and it will lead to the post condition i == n.

Inductive Step for k = 2

```
integer myfib(integer n) {
  integer c = *, p = *, i = 2;
  {
    assume( c == fib(i) );
    integer t = p + c;
    p = c;
    c = t;
    i++;
  }
  {
    assume( c == fib(i) );
    integer t = p + c;
    p = c;
    c = t;
    i++;
  }

  assert( c == fib(i) );
}
```

We get the candidate invariant by substituting i for n in the post condition.

The candidate invariant will only works for k >= 2 because it will take at least two loop unwindings to know all the needed values of the variables.

Mikhail Yasha Ramalho Gadelha
Email: myrg1g14@soton.ac.uk
Postgrad OpenDay 15th June 2015