

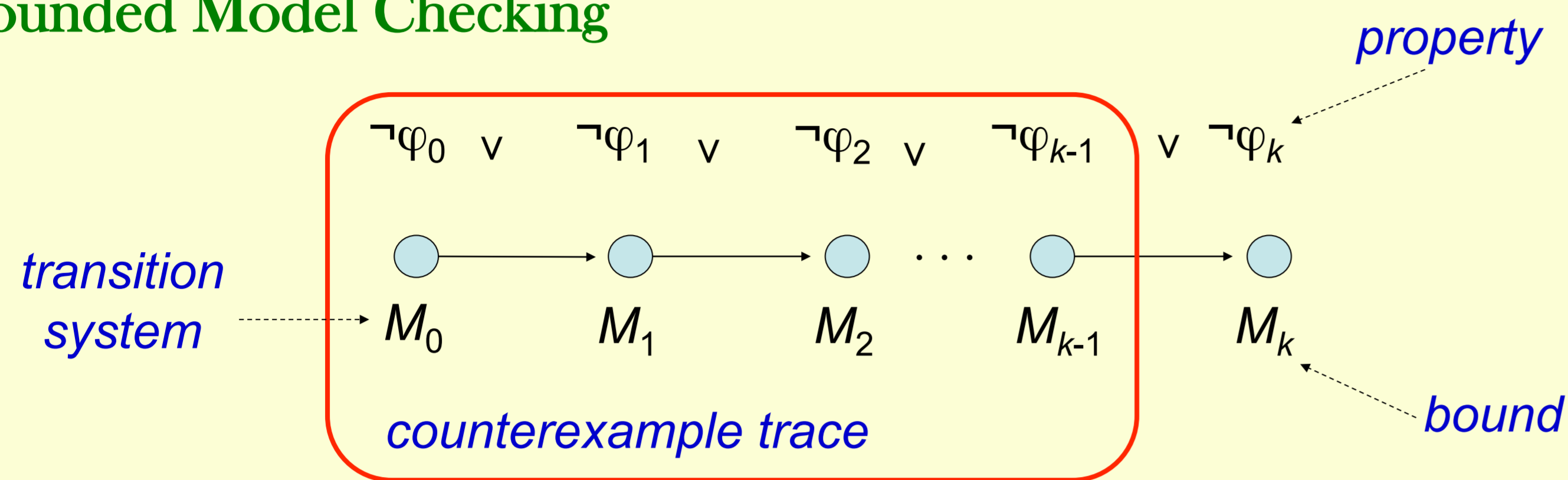
# Bounded Model Checking of C++ Programs Based on the Qt Framework

Felipe R. M. Sousa, Lucas C. Cordeiro, and Eddie B. de Lima Filho  
Electronic and Information Research Centre  
Federal University of Amazonas

## I. Introduction

- The present work identifies the main Qt features used in real applications and, based on that, creates an operational model, which provides a way to analyse and check properties related to those features.

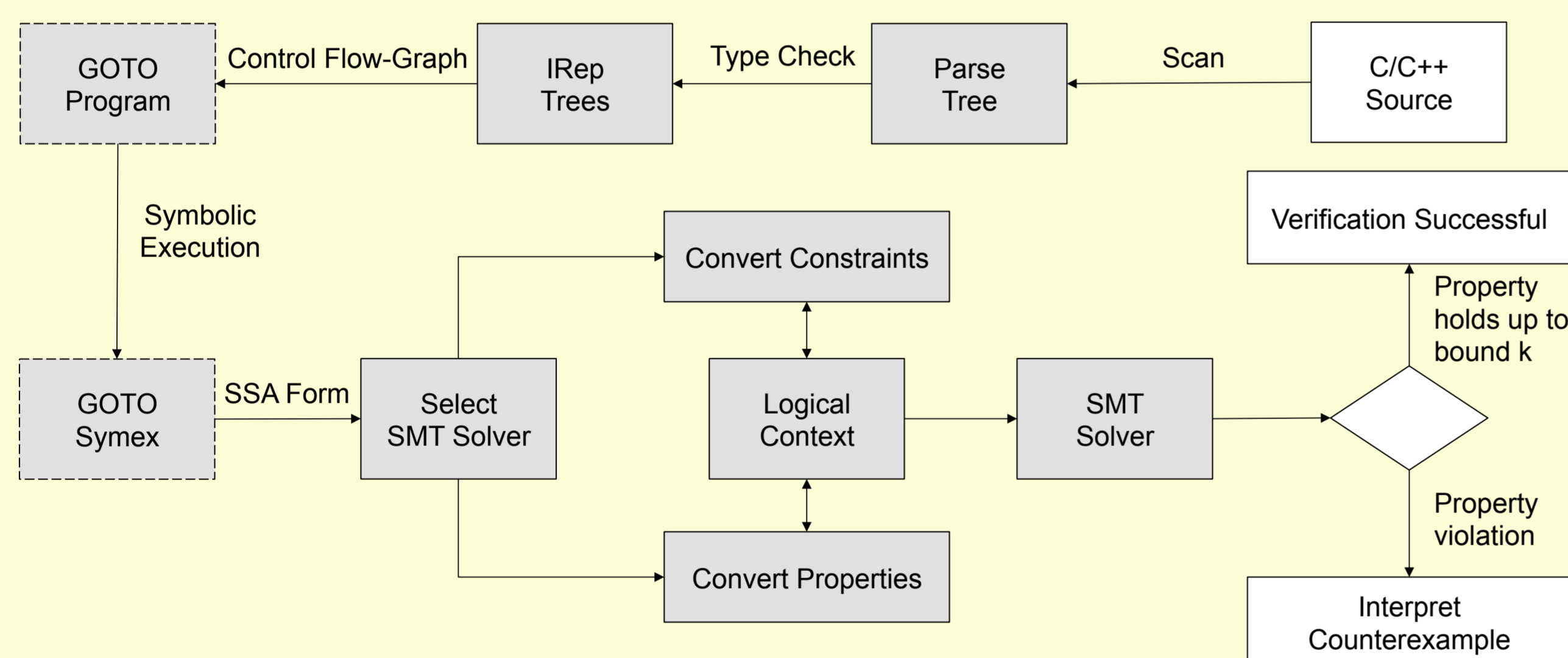
### Bounded Model Checking



Translated into a VC  $\psi$  such that:  $\psi$  is satisfiable iff  $\phi$  has counterexample of max. depth  $k$

### ESBMC++

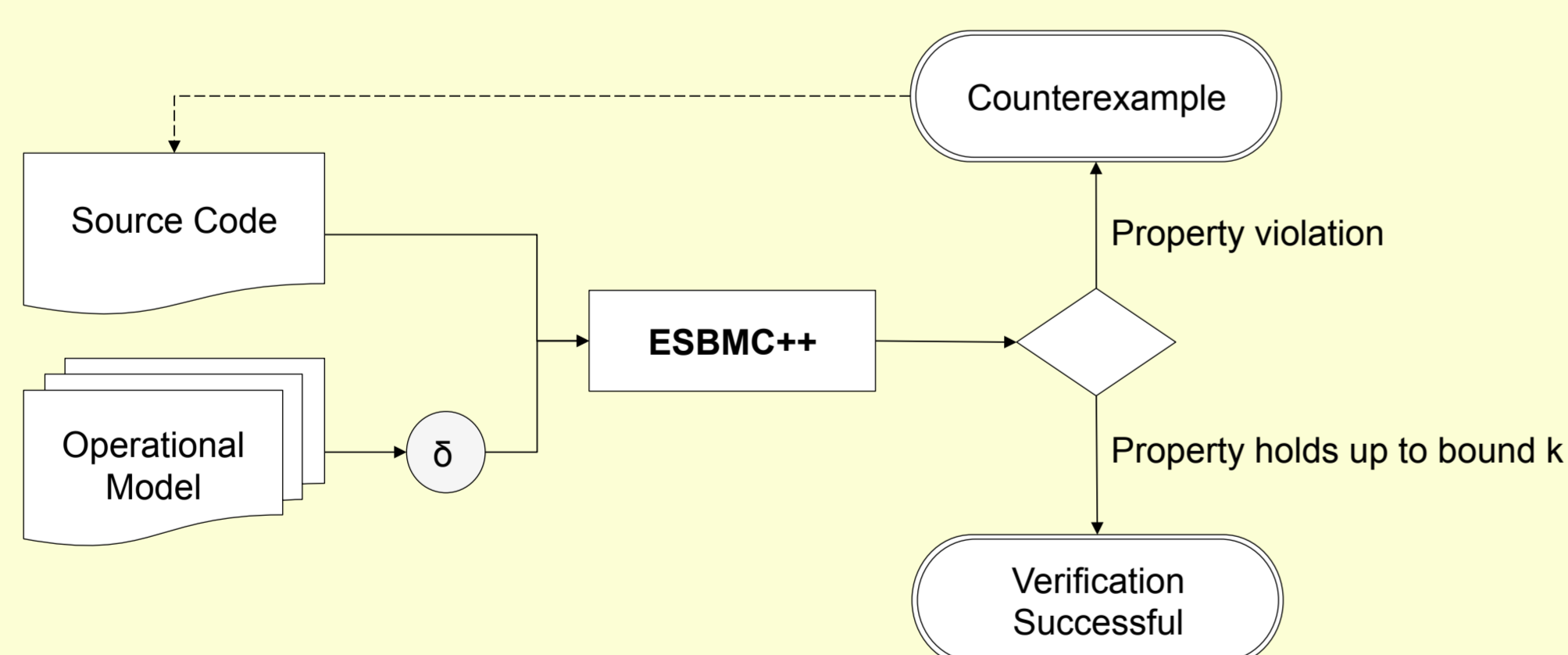
- ESBMC++ is a bounded model checker based on SMT solvers, which is used for ANSI-C/C++ single- and multi-threaded programs. Properties checked:
  - arithmetic under- and overflow, division by zero, out-of-bounds index, pointer safety, deadlocks, and data races, and assertions defined by user.



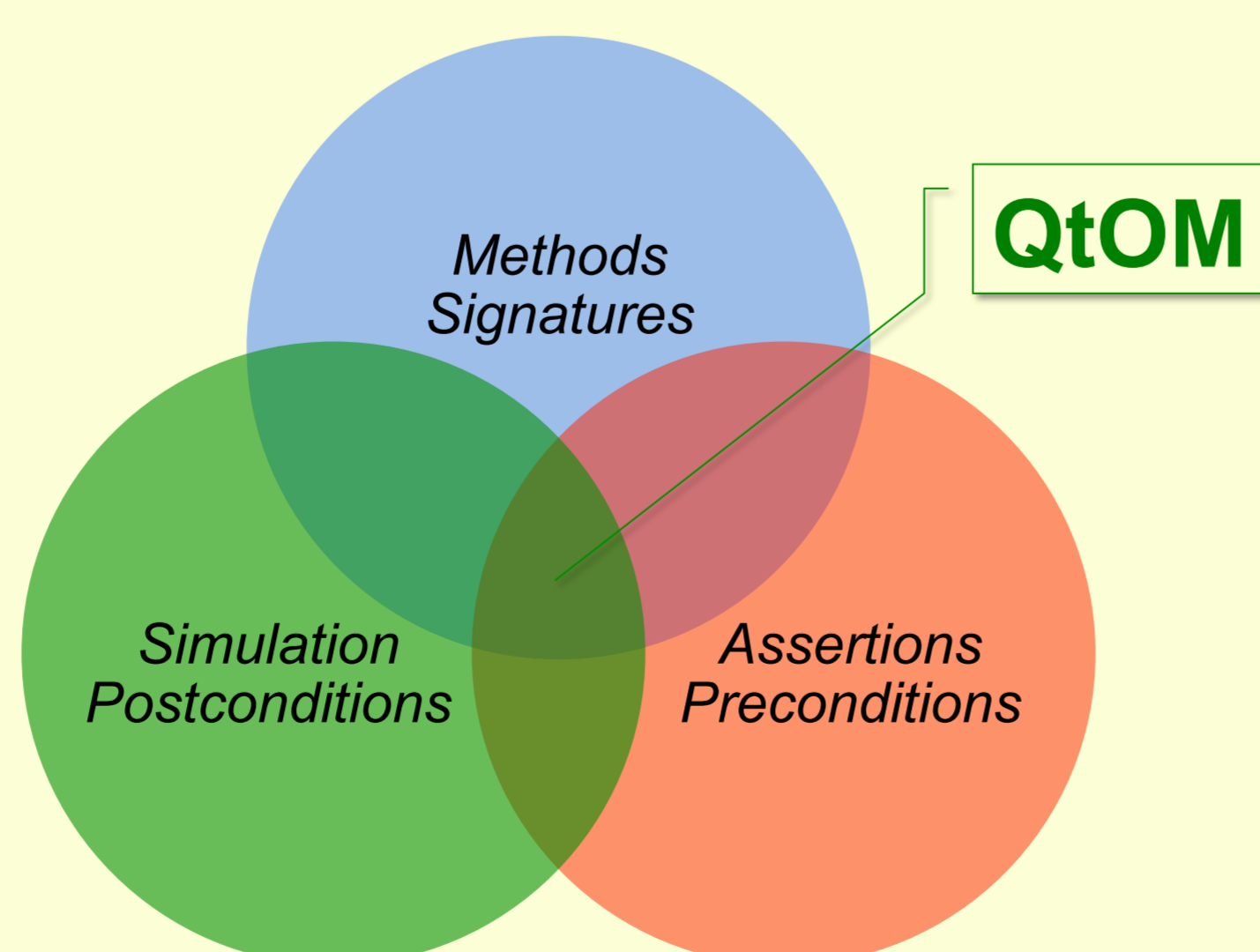
### Proposal

Apply bounded model checking to verify C++/Qt programs

## II. Verifying C++ Programs Based on the Qt Cross-Platform Framework



- Based on the framework documentation, the operational model was developed (QtOM), which considers the structure of each library and its classes, including attributes, method signatures, and function prototypes.



For further information, publications, and downloads, see:

<http://www.esbmc.org/>

Part of the results presented in this paper were sponsored by Samsung Eletrônica da Amazônia Ltda. under the terms of Brazilian federal law No. 8.387/91 (SUFRAMA).



### Pre-conditions

- A condition that must be fulfilled before a function or method can be executed.
  - Checked through assertions in the operational model;
  - When the a method/function is called, ESBMC++ interprets its behaviour as implemented in the operational model.

### Post-conditions

- An assertion that characterizes the state of the program immediately after execution of a certain function or method.

```
#include <QList>
#include <cassert>
...
int main ()
{
    ...
    QList<int> mylist;
    mylist.push_front(300);
    assert(mylist.front() == 300);
    mylist.push_front(200);
    assert(mylist.front() == 200);
    ...
}
```

### Code Fragment

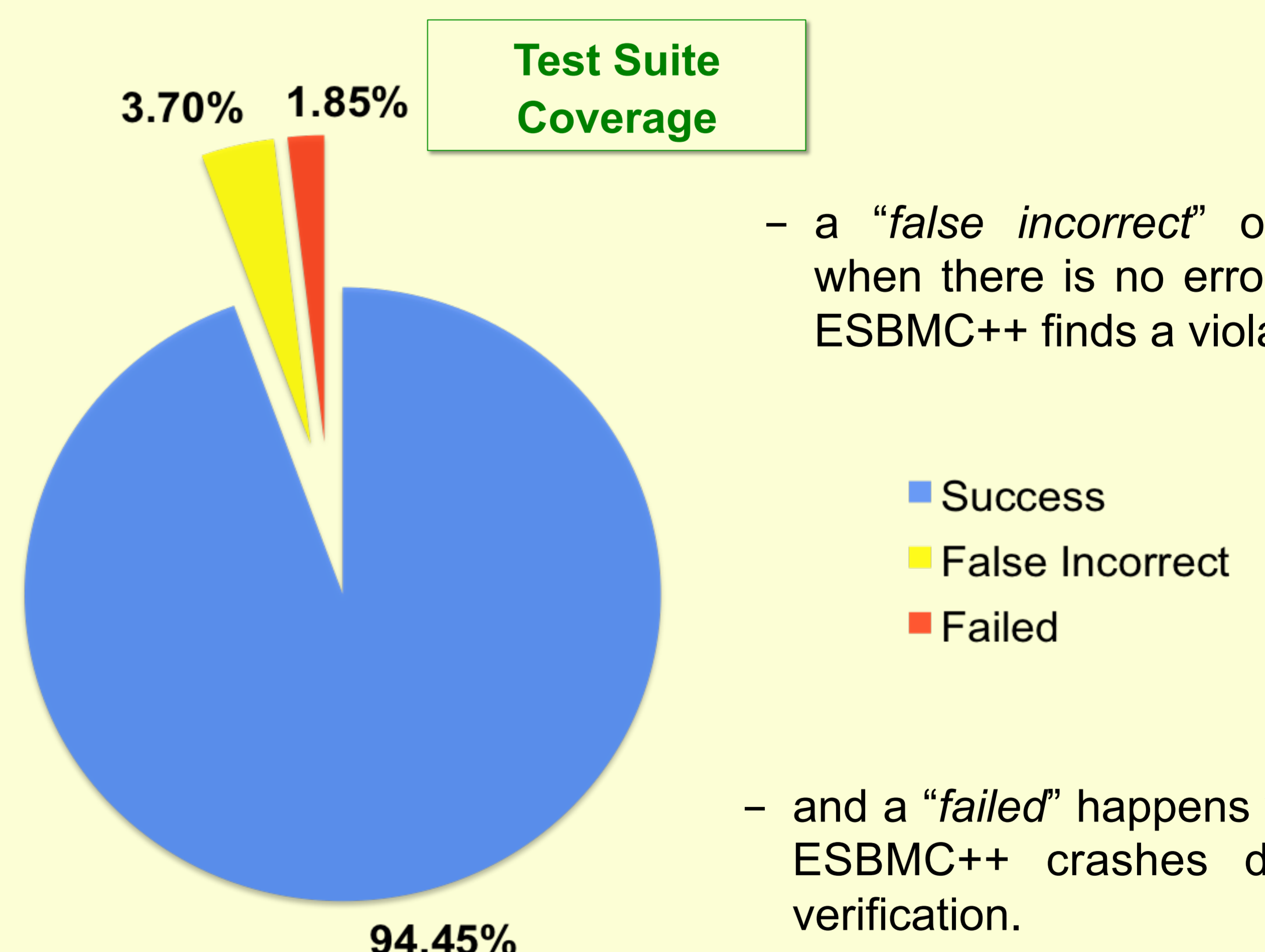
### Operational Model

```
template<class T>
class QList {
    ...
    void push_front( const value_type& x ){
        if(this->_size != 0) {
            for(int i = this->_size -1; i > -1; i--)
                this->_list[i+1] = this->_list[i];
        }
        this->_list[0] = x;
        this->_size++;
    }
    T& front() {
        __ESBMC_assert(!isEmpty(),
            ``The list must not be empty``);
        return this->_list[0];
    }
    ...
}
```

- One needs to simulate the behaviour of a certain method to consistently verify properties related to the manipulation or storage of values in a structure.

## III. Experimental Evaluation

- Currently, *esbmc-qt* test suite contains 52 benchmarks, which take about 48 seconds to be verified.



a "false incorrect" occurs when there is no error and ESBMC++ finds a violation.

and a "failed" happens when ESBMC++ crashes during verification.

## CONCLUSIONS

- This paper proposes an approach to verify C++/Qt programs using an operational model.
- The experimental results show the efficiency of this approach for verifying Qt programs and present, for the developed test suite, a success rate of 94.45%.
- As future work, more classes and libraries will be integrated into the operational model, in order to increase Qt framework coverage and validate its properties.