

2025. május 28.

A jövő Te vagy! - Kiválósági programok a Műegyetemen

Verifikációs transzformációs lépések vizsgálata és fejlesztése

Formális módszerek

Ösztöndíjas neve: Somorjai Márk István

Konzulens: Dr. Vörös András

Pályázati azonosító: EKÖP-24-2-BME-220

Formal Verification

- Safety Critical Systems
 - correctness is crucial, e.g. planes, nuclear reactors, cars
- Formal verification: mathematical proof/refutation to safety
 - e.g. error reachability, no overflows
- Ability to detect bugs earlier during development
 - reduce development time and cost
 - improve code quality
- 2 main approaches: model checking and deductive verification

Abstraction-Based Model Checking

Program

CFA

ARG

(Control Flow Automaton)

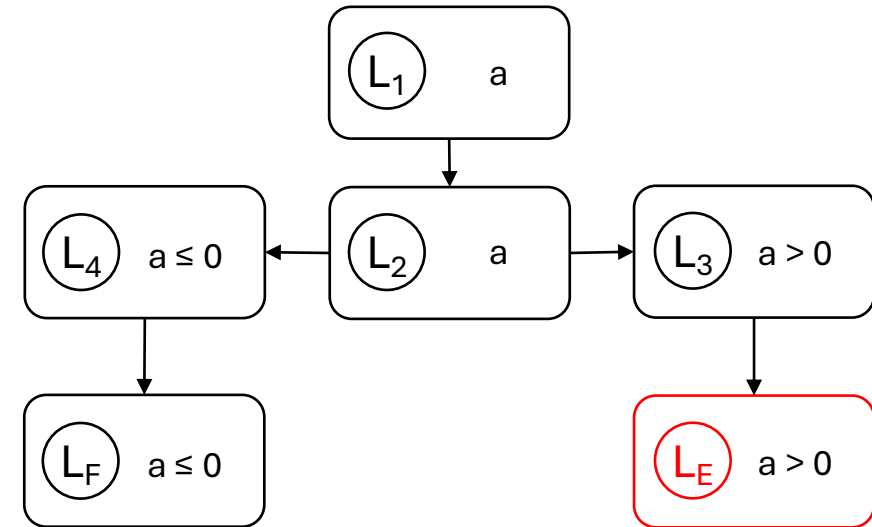
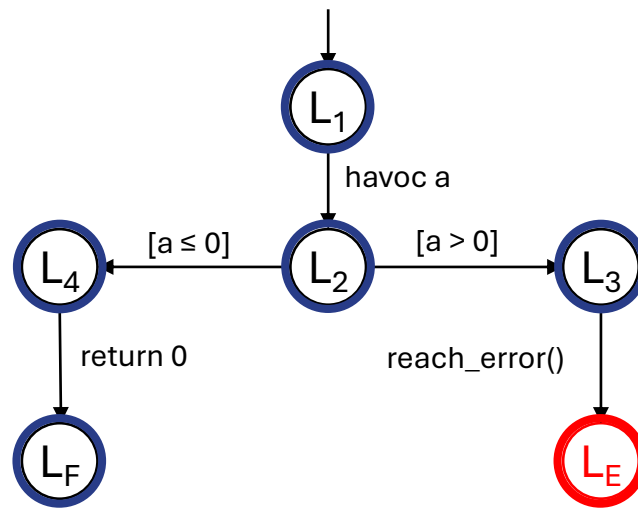
(Abstract Reachability Graph)

locations
statements

nodes
edges

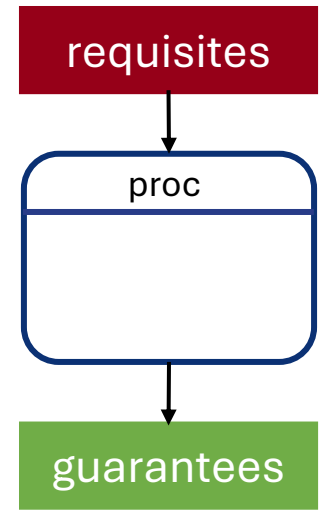
abstract states
transitions

```
int main() {  
  int a = input();  
  if (a > 0) {  
    reach_error();  
  }  
  return 0;  
}
```



Deductive Verification

- Proving the correctness of programs
 - correctness → mathematical statements
 - proof in theorem provers
 - ✓ correct
 - ✗ incorrect
 - ? unknown
- Contracts: specify behavior of software components
 - requirements on inputs
 - guarantees on outputs
- Verification with contracts:
 1. Verify each component's conformity
 2. Use contracts to verify safety



modularization
more efficient

Comparison of Verification Methods

Deductive Verification

- more efficient (modularized), but
- requires manual specification
 - engineering effort
 - specialized knowledge

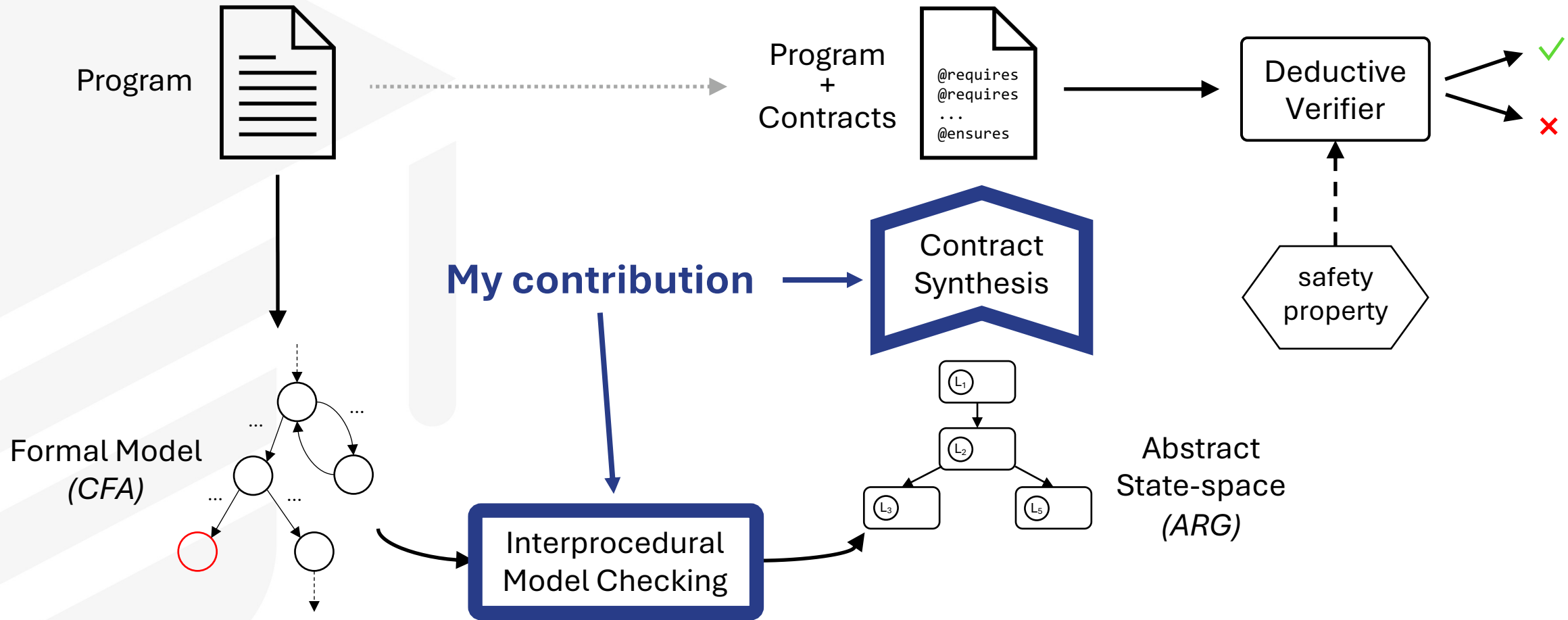
Model Checking

- less efficient, but
- **automated**

My work:

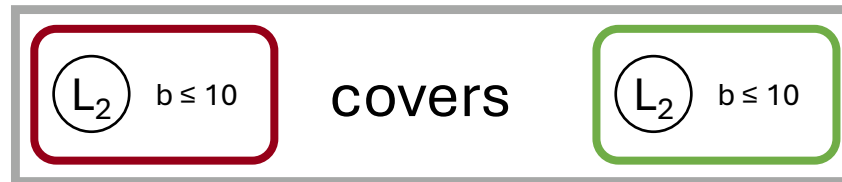
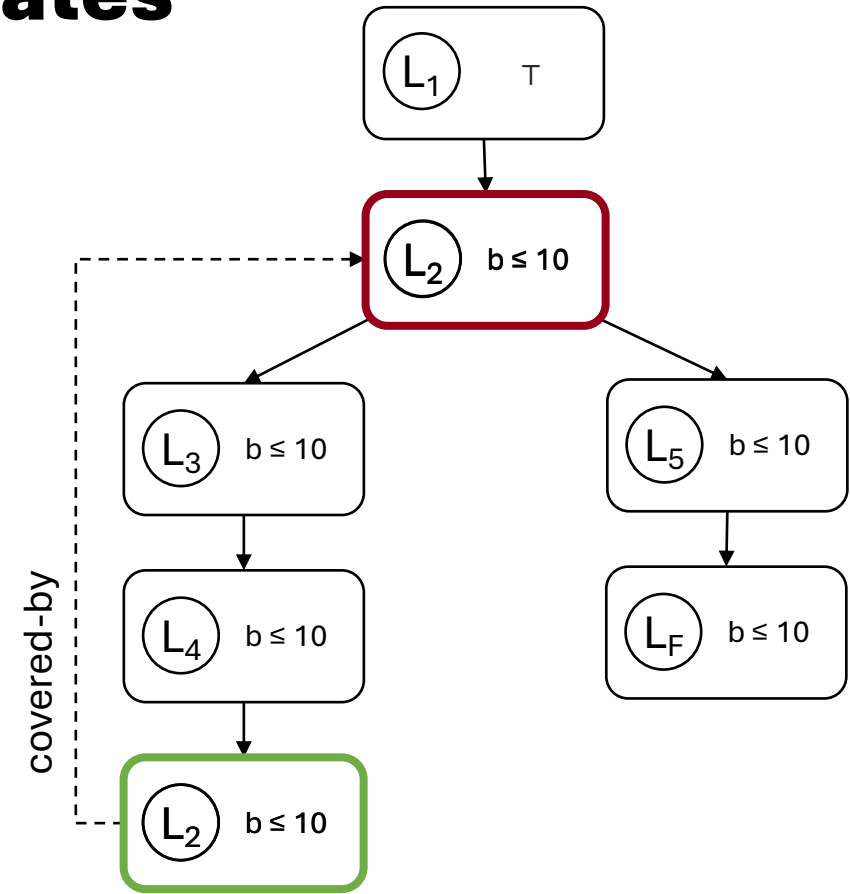
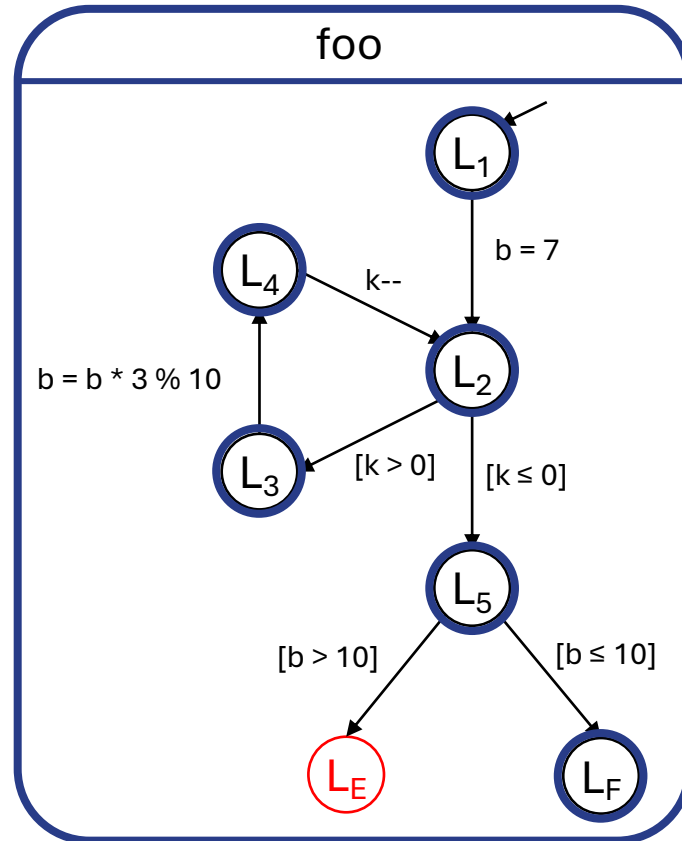
Synthesize contracts from the abstract state-space of programs, using abstraction-based, **interprocedural model checking**, in order to enable **automated deductive verification**.

Automated Contract Synthesis

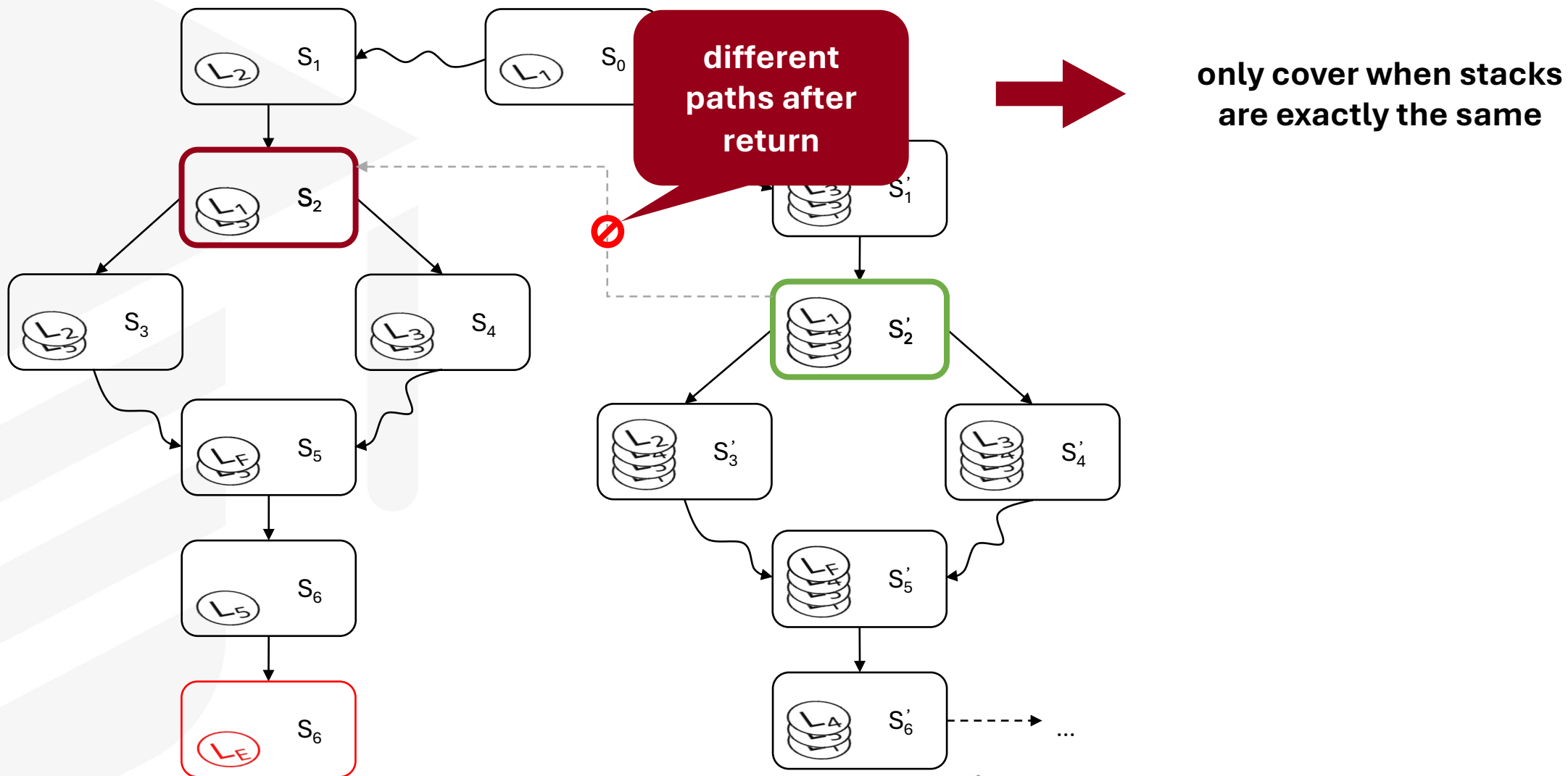


Covering Abstract States

```
void foo(int k) {
  int b = 7;
  while (k > 0) {
    b = b * 3 % 10;
    k--;
  }
  assert(b ≤ 10);
}
```

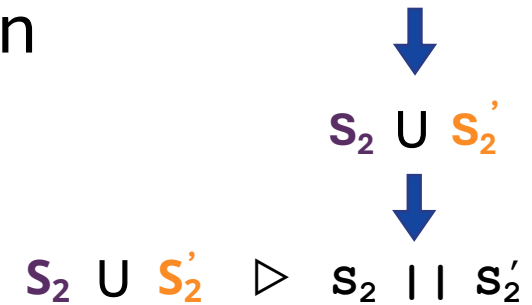
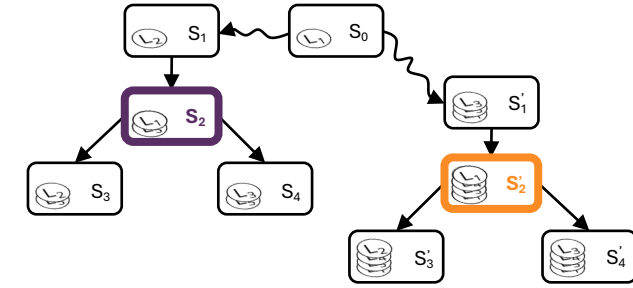


Interprocedural Covering



Contract Synthesis

1. Explore abstract state-space of program
 - tracked predicates chosen by some heuristic
2. Collect abstract states in initial locations of procedures
3. Combine abstract data states in the same location
4. Synthesize contracts from data states
 - translation ($\triangleright$) from abstract domain to contracts
5. Add contracts to program



```
/* @requires   $S_2 \parallel S'_2$  */  
void proc(...) {  
  ...  
}
```

Example

5.

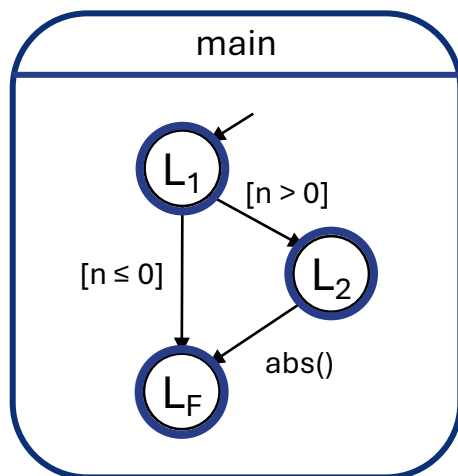
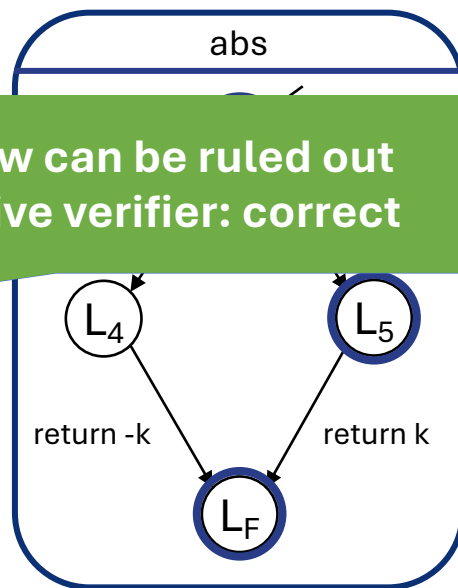
```

/* @requires
   k >= 0 */
int abs(int k) {
  if (k < 0) {
    return -k;
  } else {
    return k;
  }
}

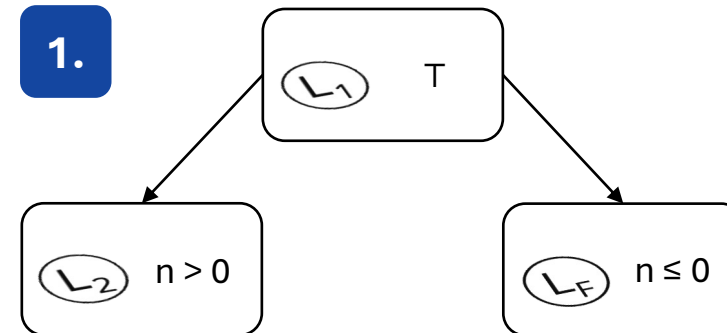
int main() {
  int n = input();
  if (n > 0) {
    abs(n);
  }
}

```

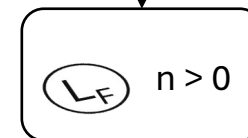
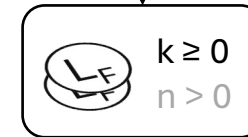
overflow can be ruled out
deductive verifier: correct



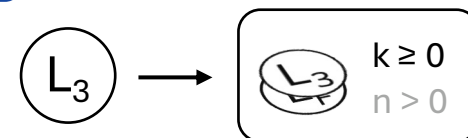
1.



2.

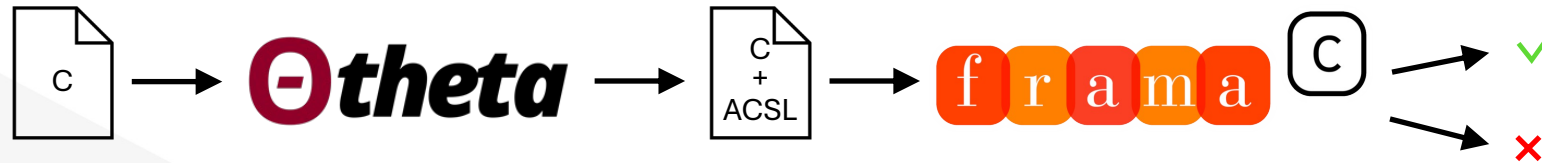


3-4.



$k \geq 0 \triangleright k \geq 0$

Evaluation



- Input: C programs from the SV-COMP benchmark repository¹
 - 1213 tasks for reachability
 - 1625 tasks for overflow
- Contract synthesis implementation in Theta²
 - ANSI C Specification Language³ (ACSL) contracts generated
- Frama-C⁴: open-source verification framework
 - deductive verification – used for reachability
 - value analysis – used for overflow
- Using BenchExec, # solved tasks (15-minute timeout) is shown

¹ <https://gitlab.com/sosy-lab/benchmarking/sv-benchmarks>

² <https://github.com/ftsrg/theta>

³ <https://frama-c.com/html/acsl.html>, ⁴ <https://frama-c.com>

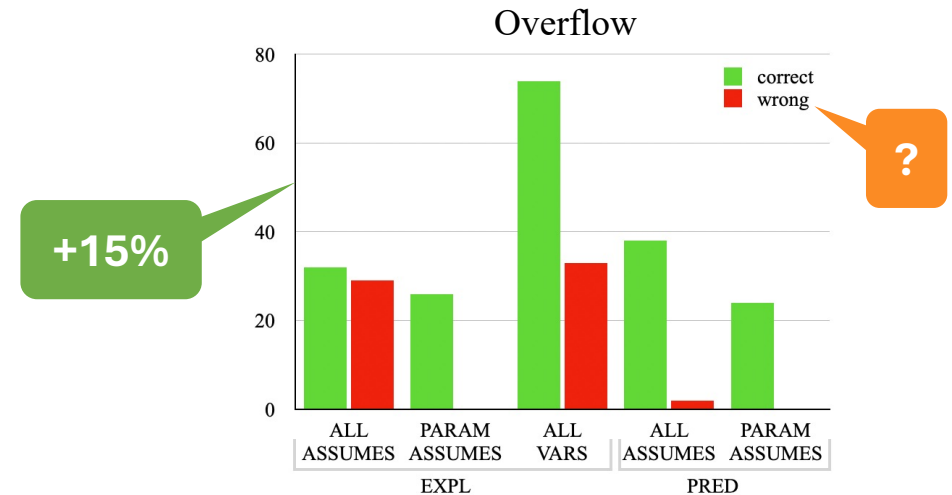
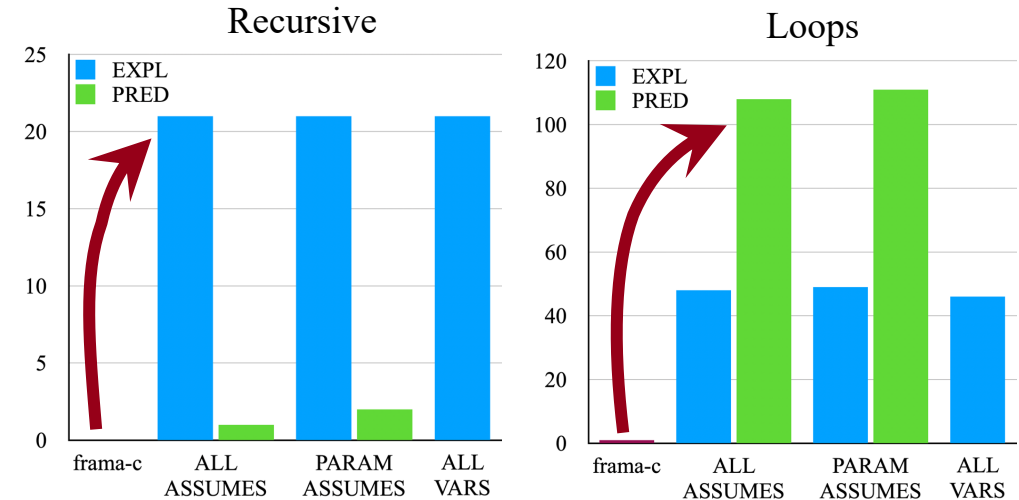
Benchmark Results

- Reachability

- recursive, loops: contracts enable Frama-C to verify programs
- other tracks: Frama-C still unable to verify any programs
- toolchain solved tasks that Theta was not able to alone

- Overflow

- 101 new tasks combined (+15%)
- wrong results in certain configs
 - but Frama-C does that by itself too



Summary

- Research results: novel contract synthesis technique
 - abstract state-space $\longrightarrow$ contracts
 - interprocedural extensions to model checking
 - benchmark campaign: approach automates deductive verification
- Results available to the institution – ✓ implementation open-source¹
- Present work at Scientific Students' Association Conference – ✓ I. place²
- Present work at international conference – ✓ FMCAD '24 Student Forum³
- Present work at University Excellence Scholarship Program Conference – ✓
- Teaching Assistance and Tutoring – ✓ Declarative Programming

¹ <https://github.com/ftsrg/theta>

² <https://tdk.bme.hu/conference/VIK/2024/sessions/sw/paper/Automatikus-kontraktus-szintezis-deduktiv>

³ <https://doi.org/10.34727/2024/isbn.978-3-85448-065-5>