AProVE (KoAT + LoAT)

Automatic Termination Analysis of C Programs

Nils Lommen, Florian Frohn, and Jürgen Giesl

Overview

- AProVE (KoAT + LoAT) [1] is a framework to analyze termination of **C** Programs
- Programs are transformed into *Integer Transition Systems (ITSs)*
- ITSs are analyzed by our tools KoAT [2] and LoAT [3]

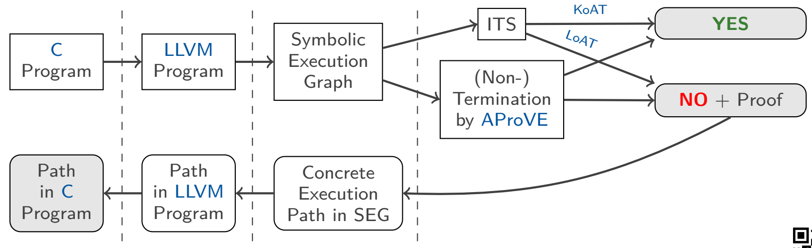


Figure 1: AProVE (KoAT + LoAT) for (Dis)proving Termination

Exemplary C Program

Does the following program terminate?

```
extern int _nondet(void);

int main() {
  int x = _nondet();
  int y = _nondet();

  while(x < y) {
    x = 3*x;
    y = 2*y;
  }
  return 0;
}
```

LLVM Program

- **C** program is compiled into **LLVM** code using Clang.
- **LLVM** fragment of the loop body:

```
%10 = load %1          # load x
%11 = mul 3 %10        # multiply x by 3
store %11, %1          # store x
%12 = load %2          # load y
%13 = mul 2 %13        # multiply y by 2
store %13, %2          # store y
br %6                  # jump to loop guard
```

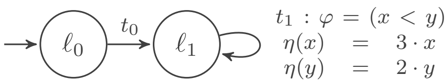
Symbolic Execution Graph (SEG) & ITS

SEG represents all possible program runs, augmented with invariants:

- Its nodes are *abstract states* that represent sets of actual program states
- SEG handles the heap, pointer arithmetic, and recursive data structures
- **LLVM** code is transformed automatically into an SEG

ITSs are a simple language for integer programs:

- Turing-complete formalism with only integer variables over $\mathbb{Z}$
- SEG is transformed into ITS



KoAT (Termination & Upper Time Bounds)

- Automated complexity and termination analysis of ITSs
- Alternating modular inference of runtime and size bounds
- How *often* can a transition be executed?
  - Multiphase Linear Ranking Functions
    ↪ Use SMT-solver Z3 to infer well-founded relation
  - TWN-Loops
    ↪ Reduce termination problem to SMT problem
    Completeness for the class of so-called TWN-loops
- How *large* are the variables?
  - Compute bounds for each change of a variable
    ↪ Over-approximate the number of changes by runtime bounds
  - Use runtime bounds and closed forms of loops

LoAT (Non-Termination and more)

Features	**Techniques**	
• *non*-termination	**ADCL**	DFS + acceleration
• *lower* time bounds	**ABMC**	BFS + acceleration
• safety / unsafety	**TRL**	BFS + recurrence analysis

Non-term. via Acceleration Driven Clause Learning

- Depth-first exploration of state space
- Applies *acceleration* when a loop is encountered
  (acceleration: under-approximation of the loop's transitive closure)
- Non-term. proofs as "by-product" of acceleration
- Exploits *redundancy* to cut off infinite branches

References

[1] Nils Lommen and Jürgen Giesl. AProVE (KoAT + LoAT) Website: `https://koat.verify.rwth-aachen.de/svcomp25`.

[2] Nils Lommen, Éléanore Meyer, and Jürgen Giesl. KoAT Website: `https://koat.verify.rwth-aachen.de/`.

[3] Florian Frohn and Jürgen Giesl. LoAT Website: `https://loat-developers.github.io/LoAT/`.