AProVE: Automated Program Verification Environment

Jürgen Giesl

LuFG Informatik 2, RWTH Aachen University, Germany

joint work with

C. Aschermann, M. Brockschmidt, F. Emmes, S. Falke, F. Frohn, C. Fuhs, M. Hark, J. Hensel, M. Naaf, L. Noschinski, P. Schneider-Kamp, T. Ströder, ...
language-specific features when generating symbolic execution graph

back-end analyzes Integer Transition Systems and/or Term Rewrite Systems

powerful termination analysis

- Termination Competition since 2004 (Java, C, Haskell, Prolog, TRS)
- SV-COMP since 2014 (C)
AProve for Complexity Analysis

Symbolic Execution Graph

- Java
- C
- Haskell
- Prolog

Worst-Case Upper Bounds
- Best-Case Lower Bounds
- Worst-Case Lower Bounds

Front-End

Back-End

Why worst-case lower bounds?
- tight bounds
- detect bugs
- detect potential attacks (DoS)

Why worst-case lower bounds?

 Martial input size

 runtime
AProVE for Complexity Analysis

**Java:**
- adapt transformation of Java to ITSs for upper bounds (iFM’17)

**C:**
- upper bounds for bitvector programs (JLAMP’18)

**Haskell:**

**Prolog:**
- infer upper bounds for Prolog from complexity of TRSs (PPDP’12)

**Symbolic Execution Graph**

**ITS:**
- alternate inference of size and time upper bounds (TACAS’14, TOPLAS’16)
- lower bounds by adapting ranking functions (IJCAR’16)

**TRS:**
- upper bounds for innermost rewriting by dep. pairs (CADE’11, JAR’13)
- use upper innermost bounds also for full rewriting (LPAR’17)
- semi-decision procedure for constant upper bounds (IPL’18)
- infer upper bounds for TRSs by ITSs (FroCoS’17)
- lower bounds by induction or syntactic criteria (RTA’15, JAR’17)

**KoAT**

**Worst-Case Upper Bounds**

**LoAT**

**Worst-Case Lower Bounds**