Florian Frohn

Automatic Complexity Analysis of Programs

Supervisor: Jürgen Giesl

LuFG Informatik II

RWTH AACHEN UNIVERSITY

APROVE

runtime

Term Rewrite System (TRS) $\mathcal{R}$

$\mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{plus}(x, \mathsf{s}(y))$
$\mathsf{plus}(0, y) \to y$

Integer Transition System (ITS) $\mathcal{I}$

$\mathsf{plus}(x, y) \to \mathsf{plus}(x-1, y+1) \mid x > 0$
$\mathsf{plus}(x, y) \to y \mid x = 0$

Java Program $\mathcal{J}$

```
int plus(int x, int y) {
  while (x > 0) {
    x--; y++;
  }
  return y;
}
```

Java $\to$ ITS

- execute program symbolically
  $\curvearrowright$ symbolic evaluation graph
- translate graph to ITS
- apply techniques for ITSs
  - translates $\mathcal{J}$ to $\mathcal{I}$

$\curvearrowright \mathcal{O}(x)$ **iFM '17**

Recursive ITS $\to$ ITS

- analyze non-recursive sub-ITSs independently
- approximate runtime and size of result
- replace calls to analyzed program parts with obtained bounds
- currently restricted to $\mathbb{N}$ **FroCoS '17**

TRS $\to$ Recursive ITS

- goal: analyze complexity w.r.t. eager evaluation
- translate TRS to recursive ITS (RITS) over $\mathbb{N}$
- apply techniques for recursive ITSs
  - translates $\mathcal{R}$ to $\mathcal{I}$

$\curvearrowright \mathcal{O}(x)$ **FroCoS '17**

LoAT

- adapt ranking func. for lower bounds $\curvearrowright$ metering func.
  - x is metering func. for plus
- compute resulting variable values via recurrence solving
  - $y' = y + x$ after x iterations
- accelerate loops
  - $\mathsf{plus}(x, y) \to \mathsf{plus}(0, y+x) \mid x > 0$

$\curvearrowright \Omega(x)$ **IJCAR '16**

Strategy Switching

- goal: analyze complexity w.r.t. unrestricted evaluation strategy
- prove that eager evaluation is worst
- apply techniques for eager evaluation
  - eager evaluation is worst for plus

$\curvearrowright \mathcal{O}(x)$ **LPAR '17**

Loop Detection

- syntactic criterion for linear & exponential lower bounds
  - $\mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{plus}(x, \mathsf{s}(y))$ is a *decreasing loop*

$\curvearrowright \Omega(x)$ **JAR '17**

Automated Induction Technique

- automatically "guess" conjecture
  - $\mathsf{plus}(\mathsf{s}^n(0), \mathsf{s}^m(0)) \to \mathsf{s}^{n+m}(0)$
- prove it via induction
- infer lower bound from proof

$\curvearrowright \Omega(n)$ **RTA '15**

input size

Key

- - - - - - worst case upper bound
——— worst case runtime
. . . . . . worst case lower bound

Applications

- DARPA STAC program [4]
- AProVE [1] and LoAT [3] are crucial parts of the CAGE toolchain [2]
- used to find or prove the absence of DoS vulnerabilities in Java programs

[1] AProVE – http://aprove.informatik.rwth-aachen.de
[2] CAGE – http://www.draper.com/news/draper-s-cage-could-spot-code-vulnerable-denial-service-attacks
[3] LoAT – https://github.com/aprove-developers/LoAT
[4] Space/Time Analysis for Cybersecurity (STAC) – https://www.darpa.mil/program/space-time-analysis-for-cybersecurity