

# Program Termination

<http://d3s.mff.cuni.cz>

Department of  
Distributed and  
Dependable  
Systems



*Pavel Parízek*



FACULTY  
OF MATHEMATICS  
AND PHYSICS  
Charles University

# Introduction

- Task: determine correctly whether a given program will **always finish** (for all inputs)
  - or give the answer “*unknown*”
- Practical motivation
  - Infinite execution of some loop may result in a non-responsive operating system
  - Common userspace applications get often stuck

- B. Cook, A. Podelski, and A. Rybalchenko.  
**Proving Program Termination.**  
Communications of the ACM, 54(5), 2011.
- B. Cook, A. Podelski, and A. Rybalchenko.  
**Termination Proofs for Systems Code.** PLDI 2006, ACM.

# Example program

```
1  x := input()
2  y := input();
3  while x > 0 and y > 0 do
4      if input() = 1 then
5          x := x - 1;
6          y := y + 1;
7      else
8          y := y - 1;
9      fi
10 done
```

Taken from B. Cook, A. Podelski, and A. Rybalchenko.  
Proving Program Termination. Comm. of the ACM, 54(5), 2011.

# Terminology

- Transition relation  $R$ 
  - $R \subseteq S \times S, (s, s') \in R$  iff  $s \rightarrow s'$
- Termination argument
- Well-order relation
- Ranking function  $f$
- Well-founded relation
  - $T = \{ (s_1, s_2) \mid f(s_1) > f(s_2) \}$
- Disjunctive termination argument
  - $T = T_1 \cup T_2 \cup \dots \cup T_N$

# Current state of the art

- What can be proved (disproved)
  - Famous complex problems: Ackermann's function
  - Industrial examples: Windows device drivers
    - Prover: **Terminator (T2)**
  - Sequential programs that use arithmetic expressions
- Research challenges
  - More complex programs (dynamic allocation, threads)
  - Processing non-linear arithmetic operators ( $*$ ,  $/$ ,  $%$ )
- Other applications: checking liveness properties

# Terminator (T2)

- Termination prover
  - Developed by Microsoft Research (B. Cook et al.)
- How it works
  - Iterative proving based on abstraction refinement
- Source code freely available
  - <http://mmjb.github.io/T2/>
- Implementation languages: F#, ML

- Simple language and program verifier
- Important features
  - Contracts: precondition, postcondition, invariant
  - Program termination analysis
- Resources
  - <https://github.com/dafny-lang/dafny>
  - <http://dafny.org/dafny/>



# Dafny: Example 1

```
method Compute(x: int, y: int, z: int) {  
  var x1: int := x;  
  var y1: int := y;  
  while (x1 > 0 && y1 > 0) {  
    if (z == 1) {  
      x1 := x1 - 1;  
      y1 := y1 + 1;  
    }  
    else {  
      y1 := y1 - 1;  
    }  
  }  
}
```

# Dafny: Example 2

```
method Compute(n: int) {  
  var x: int := 0;  
  while (x < n)  
  {  
    x := x + 1;  
  }  
}
```