ISR 2017 Advanced Course

Proving Program Termination via Term Rewriting

Carsten Fuhs

http://www.dcs.bbk.ac.uk/~carsten/

—Abstract—

1 Topic

Termination is the property of a program that regardless of the input, execution of the program will always come to a halt eventually. Although this property is undecidable, since the early 2000s fully automated techniques and tools for termination analysis have flourished, as witnessed also by the annual International Termination Competition [7]. In particular, automated termination analysis has been a very active topic of research for term rewriting. At the same time, term rewriting allows for a convenient representation of user-defined data structures via terms. This makes term rewriting an attractive target language for termination proving of programs via translations from dedicated programming language front-ends [3, 4, 5].

The goal of the course is to present both the general approach used for such translations and to look into concrete realizations for specific programming languages. As examples, in the course we shall discuss the lazy functional programming language Haskell [3] and the object-oriented imperative language Java [5]. In both cases, the program is executed symbolically from its entry point with suitable generalization steps to obtain an over-approximation of all possible computations. A common theme for the symbolic state representations (the "abstract domains") for the different programming languages is to capture aspects of the program states that lend themselves to a representation as terms. From this program analysis, we extract a term rewrite system such that a termination proof for this term rewrite system would imply termination of the original program. This approach enables a re-use of termination provers for term rewriting also for widely used programming languages, thus applying term rewriting as an intermediate language for proving program termination.

As far as time permits, we will also briefly discuss instantiations of this approach for the programming languages Prolog [4] and C [6] as well as extensions of standard term rewriting by built-in integer arithmetic [1, 2].

The course is complemented by hands-on exercises that allow the students to perform such translations both with pen and paper and with the help of the automatic termination analysis tool AProVE [2].

2 Learning Goals

- To construct an over-approximation of all program executions for Haskell and Java programs that is geared towards termination proving via term rewriting.
- To extract term rewrite systems from this over-approximation so that termination of the term rewrite system implies termination of the original program.
- To obtain an intuition for the choice of abstract domain for a given programming language for such over-approximations.

3 Prerequisites

Familiarity with the basics of term rewriting. Some familiarity with functional and object-oriented programming.

References

[1] Carsten Fuhs, Jürgen Giesl, Martin Plücker, Peter Schneider-Kamp, and Stephan Falke. Proving termination of integer term rewriting. In *Proc. 20th International Conference on Rewriting Techniques and Applications (RTA '09)*, volume 5595 of *Lecture Notes in Computer Science*, pages 32–47, 2009.

[2] Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Analyzing program termination and complexity automatically with AProVE. *Journal of Automated Reasoning*, 58(1):3–31, 2017. AProVE is available for download and via a web interface at `http://aprove.informatik.rwth-aachen.de/`.

[3] Jürgen Giesl, Matthias Raffelsieper, Peter Schneider-Kamp, Stephan Swiderski, and René Thiemann. Automated termination proofs for Haskell by term rewriting. *ACM Transactions on Programming Languages and Systems*, 33(2):7:1–7:39, 2011.

[4] Jürgen Giesl, Thomas Ströder, Peter Schneider-Kamp, Fabian Emmes, and Carsten Fuhs. Symbolic evaluation graphs and term rewriting – A general methodology for analyzing logic programs. In *Proc. 14th International Symposium on Principles and Practice of Declarative Programming (PPDP '12)*, pages 1–12, 2012.

[5] Carsten Otto, Marc Brockschmidt, Christian von Essen, and Jürgen Giesl. Automated termination analysis of Java Bytecode by term rewriting. In *Proc. 21st International Conference on Rewriting Techniques and Applications (RTA '10)*, volume 6 of *Leibniz International Proceedings in Informatics*, pages 259–276, 2010.

[6] Thomas Ströder, Jürgen Giesl, Marc Brockschmidt, Florian Frohn, Carsten Fuhs, Jera Hensel, Peter Schneider-Kamp, and Cornelius Aschermann. Automatically proving termination and memory safety for programs with pointer arithmetic. *Journal of Automated Reasoning*, 58(1):33–65, 2017.

[7] *Termination Competition.* `http://termination-portal.org/wiki/Termination_Competition`.