ISR 2020 Advanced Course Proposal

Automated Complexity Analysis for Term Rewriting

Carsten Fuhs

Abstract

Complexity analysis for term rewriting aims to infer bounds on the length of the longest rewrite sequence as a function of the size of the considered start terms. In this course, we look into static analysis techniques that can be used as ingredients for automatic complexity analysis tools for term rewriting. Such tools take as input a term rewrite system and provide as output asymptotic upper or lower worst-case complexity bounds (e.g., $\mathcal{O}(n^2)$) for this term rewrite system.

1 Outline

Complexity analysis for term rewrite systems (TRSs) investigates the length of the longest rewrite sequence (or: longest derivation) as a function of the size of its start term. If the set of start terms is not restricted, this complexity function is known as the *derivational complexity* of the TRS [11]. It is in general not computable; that is why the development of techniques and tools has focused on sufficient criteria for the inference of (not necessarily tight) upper and lower asymptotic bounds of the complexity function.

Research in this area initially dealt with upper bounds on the derivational complexity induced by termination proof techniques: if termination of a given TRS is proved using a particular technique, this implies that the derivational complexity of the TRS cannot exceed a bound specific to the proof technique and its parameters [11].

However, from a program analysis perspective, derivational complexity is not a suitable complexity measure: even a simple TRS with the two rules $\mathsf{double}(\mathsf{0}) \to \mathsf{0}$ and $\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$ to double a natural number has exponential derivational complexity. The issue is that derivational complexity considers start terms of arbitrary shape; in particular, start terms with nested defined symbols like $\mathsf{double}(\mathsf{double}(\ldots \mathsf{double}(\mathsf{s}(\mathsf{0}))\ldots))$ are allowed and may cause exponential derivational complexity. In contrast, computing the double of a *natural number* $\mathsf{double}(\mathsf{s}(\ldots \mathsf{s}(\mathsf{0})\ldots))$ (intuitively: applying the function double to *data*) has linear complexity. This latter notion, where defined symbols in the start term are allowed only at its root, is significantly closer to complexity analysis for conventional programming languages. The corresponding complexity function is known as the *runtime complexity* [10] of a TRS.

In this course, we will discuss a selection of automated techniques for inferring asymptotic lower and upper bounds on derivational and runtime complexity of TRSs. For *upper* bounds, the dependency pair method from termination analysis [1] has been adapted to allow for a certain modularity in the analysis [10, 14, 3]. Further techniques to find upper bounds include transformations between different complexity problems for rewriting [5, 7] or from rewriting to programs on integer numbers [13]. As witnesses for *lower* bounds, one can find “decreasing loops” or identify a family of start terms t_n parametric in the term size n for which one can prove inductively that corresponding rewrite sequences of a certain length exist [6].

If time permits, we may also briefly look at the application of complexity analysis tools for term rewriting as backends for the complexity analysis of programming languages such as Prolog [9], OCaml [2], and Java [12].

2 Duration and Exercises

The course is designed for 2 slots of 90 minutes each. These will also include selected exercises and the possibility to work with existing complexity analysis tools such as AProVE [8] and TcT [4].

References

[1] Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1–2):133–178, 2000.

[2] Martin Avanzini, Ugo Dal Lago, and Georg Moser. Analysing the complexity of functional programs: higher-order meets first-order. In *Proc. ICFP '15*, pages 152–164, 2015.

[3] Martin Avanzini and Georg Moser. A combination framework for complexity. *Information and Computation*, 248:22–55, 2016.

[4] Martin Avanzini, Georg Moser, and Michael Schaper. TcT: Tyrolean complexity tool. In *Proc. TACAS '16*, volume 9636 of *LNCS*, pages 407–423, 2016. `http://cl-informatik.uibk.ac.at/software/tct/`.

[5] Florian Frohn and Jürgen Giesl. Analyzing runtime complexity via innermost runtime complexity. In *Proc. LPAR '17*, volume 46 of *EPiC*, pages 249–268, 2017.

[6] Florian Frohn, Jürgen Giesl, Jera Hensel, Cornelius Aschermann, and Thomas Ströder. Lower bounds for runtime complexity of term rewriting. *Journal of Automated Reasoning*, 59(1):121–163, 2017.

[7] Carsten Fuhs. Transforming derivational complexity of term rewriting to runtime complexity. In *Proc. FroCoS '19*, volume 11715 of *LNAI*, pages 348–364, 2019.

[8] 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*, 58:3–31, 2017. `http://aprove.informatik.rwth-aachen.de/`.

[9] 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. PPDP '12*, pages 1–12, 2012.

[10] Nao Hirokawa and Georg Moser. Automated complexity analysis based on the dependency pair method. In *Proc. IJCAR '08*, volume 5195 of *LNAI*, pages 364–379, 2008.

[11] Dieter Hofbauer and Clemens Lautemann. Termination proofs and the length of derivations. In *Proc. RTA '89*, volume 355 of *LNCS*, pages 167–177, 1989.

[12] Georg Moser and Michael Schaper. From Jinja bytecode to term rewriting: A complexity reflecting transformation. *Information and Computation*, 261:116–143, 2018.

[13] Matthias Naaf, Florian Frohn, Marc Brockschmidt, Carsten Fuhs, and Jürgen Giesl. Complexity analysis for term rewriting by integer transition systems. In *Proc. FroCoS '17*, volume 10483 of *LNAI*, pages 132–150, 2017.

[14] Lars Noschinski, Fabian Emmes, and Jürgen Giesl. Analyzing innermost runtime complexity of term rewriting by dependency pairs. *Journal of Automated Reasoning*, 51(1):27–56, 2013.