The Termination and Complexity Competition

Jürgen Giesl[1](✉), Albert Rubio[2](✉), Christian Sternagel[3](✉),
Johannes Waldmann[4](✉), and Akihisa Yamada[5](✉)

[1] LuFG Informatik 2, RWTH Aachen University, Aachen, Germany
giesl@informatik.rwth-aachen.de
[2] Universitat Politècnica de Catalunya, Barcelona, Spain
albert@cs.upc.edu
[3] Department of Computer Science, University of Innsbruck, Innsbruck, Austria
christian.sternagel@uibk.ac.at
[4] Institut für Informatik, HTWK Leipzig, Leipzig, Germany
johannes.waldmann@htwk-leipzig.de
[5] NII, Tokyo, Japan
akihisayamada@nii.ac.jp

Abstract. The termination and complexity competition (`termCOMP`) focuses on automated termination and complexity analysis for various kinds of programming paradigms, including categories for term rewriting, integer transition systems, imperative programming, logic programming, and functional programming. In all categories, the competition also welcomes the participation of tools providing certifiable output. The goal of the competition is to demonstrate the power and advances of the state-of-the-art tools in each of these areas.

1 Introduction

Termination and complexity analysis have attracted a lot of research since the early days of computer science. In particular, termination for the rewriting model of computation is essential for methods in equational reasoning: the word problem [18] asks for convertibility with respect to a rewrite system, and some instances can be solved by a completion procedure where termination needs to be checked in each step [34]. Term rewriting is the basis of functional programming [42], which, in turn, is the basis of automated theorem proving [13]. As early examples for the importance of termination in other domains and models of computation we mention that completion is used in symbolic computation for the construction of Gröbner Bases

for polynomial ideals [15], and that boundedness of Petri Nets can be modeled by termination of vector addition systems, which is decidable [33].

Both termination and complexity (or resource consumption) are very relevant properties for many computation systems and keep being the focus of interest in newly emerging technologies. For instance, complexity analyzers are applied to analyze large Java programs in order to detect vulnerabilities [45].

Another particularly interesting example are smart contracts in blockchains, which are becoming very popular. Providing tools for analyzing their termination and bounding their resource consumption is critical [2]. For example, transactions that run out-of-gas in the Ethereum blockchain platform throw an exception, their effect is reverted, and the gas consumed up to that point is lost.

Deciding the (uniform) termination problem is to determine whether a given program has only finite executions for all possible inputs. Termination is a well-known undecidable property for programs written in any Turing complete language, and any complexity analyzer must include termination analysis as well. Despite this challenging undecidable scenario, powerful automatic tools for many different formalisms are available nowadays.

History of termCOMP. After a tool demonstration at the Termination Workshop 2003 (Valencia) organized by Albert Rubio, the community decided to hold an annual termination competition and to collect benchmarks in order to spur the development of tools and new termination techniques. Since 2004 the competition, known as termCOMP, has been organized annually, with usually between 10 and 20 tools participating in the different categories on termination, complexity, and/or certification. The actual organizers of the competition have been Claude Marché (from 2004 to 2007), René Thiemann (from 2008 to 2013), Johannes Waldmann (from 2014 to 2017), and Akihisa Yamada (since 2018). Recent competitions have been executed live during the main conferences of the field (at FLoC 2018, FSCD 2017, WST 2016, CADE 2015, VSL 2014, RDP 2013, IJCAR 2012, RTA 2011, and FLoC 2010). Information on all termination and complexity competitions is available from http://termination-portal.org/.

Computational resources for the execution of the competition have been provided by LRI, Université Paris-Sud (from 2004 to 2007) and by the University of Innsbruck (from 2008 to 2013). Since 2014, the competition runs on StarExec, a cross-community service at the University of Iowa for the evaluation of automated tools based on formal reasoning. It provides a single piece of storage and computing infrastructure to the communities in computational logic developing such tools [48].

From 2014 to 2017, competition results were presented using a separate web application star-exec-presenter developed at HTWK Leipzig [40], giving both an aggregated view of results, as well as detailed results per category. Additionally, it provides options for sorting and selecting subsets of benchmarks and solvers according to various criteria, as well as for comparing results of various competitions and/or test runs. This helps to estimate progress and to detect inconsistencies. Since 2018, starexec-master [50] (the successor of star-exec-presenter) is in use (see Fig. 1 in Sect. 2).

Competition Benchmarks. The benchmarks used to run the competition on are collected in the *Termination Problem Data Base* (TPDB for short), which was originally created by Claude Marché, Albert Rubio, and Hans Zantema, and later on maintained, extended, and reorganized by René Thiemann, Johannes Waldmann, and Akihisa Yamada. Many researchers have contributed with new benchmarks over the years. The current version of TPDB (10.6) contains a total of 43,112 benchmarks and extends over 674 MByte (uncompressed).

The termination competitions started with categories on termination of string rewrite systems (SRSs) and term rewrite systems (TRSs). Apart from standard rewriting, there were also categories based on adding strategies and extensions like equational, innermost, or context-sensitive rewriting. Further categories were introduced afterwards, including, for instance, higher-order rewriting (since 2010) and cycle rewriting (since 2015). Categories on complexity analysis of rewrite systems were added in 2008.

Regarding analysis tools for programming languages, a category on termination of logic programs was already part of the competition in 2004. Categories for other programming paradigms were introduced later: since 2007 there is a category for functional (Haskell) programs, since 2009 termination of Java programs is also considered, and since 2014 C programs are handled as well. Moreover, back-end languages like integer transition systems (ITSs) or integer term rewriting are part of termCOMP since 2014. Last but not least, complexity analysis categories for some of these languages have also been included recently.

Finally, the first certification categories on rewriting were included in 2007 and have been extended to some other languages and formalisms over the years.

Overview. In the remainder of this paper we will

- describe the organization of termCOMP in its 2019 edition (Sect. 2),
- give a detailed account of the categories in the used benchmark collection (Sect. 3),
- and give an overview on the tools and techniques at the previous termCOMP 2018 (Sect. 4).

2   Organization of the Competition

In 2019 we plan to run the competition on StarExec again. Each tool will be run on all benchmarks of the categories it is registered for, with a wall-clock timeout of 300 s per example. Tools are expected to give an answer in the first line of their standard output, followed by a justification for their answer.

In termination categories, the expected answers are YES (indicating termination), NO (indicating nontermination), and MAYBE (indicating that the tool had to give up). Each YES or NO answer will score one point, unless it turns out to be incorrect. Each incorrect answer scores −10 points.

In complexity categories, an answer specifies either or both upper- and lower-bound (worst-case) complexity. The score of an answer is the sum of the scores for the upper-bound and lower-bound, each of which depends on the number of

other participants. Details of the answer format and scoring scheme are available at http://cbr.uibk.ac.at/competition/rules.php.

In contrast to previous years, we will not run the competition live but before the TACAS conference takes place. We reserve about two weeks for resolving technical issues, analyzing conflicting answers, and debugging. If participants fail to agree on the treatment of conflicts, the steering committee will finally decide which answer will be penalized.

The competition results will be presented using the starexec-master web front end [50], see Fig. 1.

Termination of Rewriting

category	ranking
TRS Standard 30034	AProVE 1283; NaTT 1018; ttt2-1.17+nonreach 998; muterm 5.18 832; Wanda 619;
TRS Standard Certified 30038	AProVE 1200; ttt2-1.17+nonreach 925;
SRS Standard 30035	MultumNonMulta3.12_29June2018A 976; AProVE 971; ttt2-1.17+nonreach 744; NaTT 203; muterm 5.18 136;
SRS Standard Certified 30039	AProVE 838; ttt2-1.17+nonreach 592;
TRS Relative 30036	NaTT 62; AProVE 56; ttt2-1.17+nonreach 39;
TRS Relative Certified 30040	AProVE 50; ttt2-1.17+nonreach 42;
SRS Relative 30037	MultumNonMulta3.12_29June2018A 142; AProVE 90; ttt2-1.17+nonreach 24; NaTT 7;
SRS Relative Certified 30041	AProVE 90; ttt2-1.17+nonreach 28;
TRS Equational 30042	AProVE 64; muterm 5.18 63; NaTT 46;
TRS Equational Certified 30043	AProVE 63; NaTT 25;
TRS Conditional 30044	muterm 5.18 101; AProVE 84;
TRS Context Sensitive 30045	muterm 5.18 101; AProVE 97;
TRS Innermost 30046	AProVE 296; muterm 5.18 208;
HRS (union beta) 30047	sol 37957 219; Wanda 165; SizeChangeTool 93;

Termination of Programs

category	ranking
C 30048	AProVE 292; UltimateAutomizer 0;
C Integer 30049	AProVE 316; VeryMax-termCOMP17 315; UltimateAutomizer 0;
Integer Transition Systems 30050	VeryMax-termCOMP17 1025; irankfinder v1 524; Ctrl 450;
Integer TRS Innermost 30051	AProVE 102; Ctrl 85;

Complexity Analysis

category	ranking
Complexity: ITS 30054	AProVE 1688; CoFloCo 2018 648;
Complexity: C Integer 30055	CoFloCo 2018 518; AProVE 476;
Runtime Complexity: TRS 30091	AProVE 2209; tct 2018-07-13 1303;
Runtime Complexity: TRS Innermost 30092	AProVE 1787; tct 2018-07-13 998;
Runtime Complexity: TRS Innermost Certified 30094	tct 2018-07-13 407; AProVE 388;

Fig. 1. The starexec-master web front end summarizing the 2018 competition

3   Categories

Benchmarks are grouped in the TPDB according to the underlying computational model (rewriting or programming) and to the aim of the analysis (termination or complexity). This organization results in the following three *meta categories* since termCOMP 2014: *termination of rewriting*, *termination of programs*, and *complexity analysis*. (A further split of *complexity analysis* into two meta categories "*complexity of rewriting*" and "*complexity of programs*" might be considered in the future if there are categories concerning the complexity of several different programming languages.)

Roughly speaking, the two termination meta categories cover, on the one hand, termination of different flavors of rewriting according to various strategies (*termination of rewriting*), and on the other hand, termination of actual programming languages as well as related formalisms (*termination of programs*).

Which categories of a given meta category are actually run during a competition depends on the registered participants. Any category with at least two participants is run as part of its associated meta category. Of course, it is desirable to have as many participants as possible and therefore all developers of termination and complexity analysis tools are strongly encouraged to participate in the competition. In addition, as a special case, all those categories having only a single participant are collected into the auxiliary *demonstration* meta category. (While *demonstration* categories are not considered for computing scores and are thus not part of a competition in terms of awards or medals, this at least allows us to make unique tools visible to the outside world.)

Independent of their respective meta categories, there are several categories that come also in a special *certified* variant (marked by ☑ below). Before 2007, the standard approach of participating tools was to give some textual justification for their answers. However, there was no consensus on the format or the amount of detail for such justifications. Automated termination and complexity tools are rather complex programs. They are typically tuned for efficiency using sophisticated data structures and often have short release cycles facilitating the quick integration of new techniques. So, why should we trust such tools? *Certification* is the answer to this question. Tools that participate in certified categories are required to produce their justifications in a common format, the *certification problem format*, or CPF [46] for short. Justifications in this format are usually called *certificates*. To make sure that certificates are correct, certified categories employ a *certifier*—an automated tool that is able to rigorously validate a given certificate. For recent editions of termCOMP this certifier is CeTA [6,49], short for "certified tool assertions". Its reliability is due to the fact that its correctness has been established using the proof assistant Isabelle/HOL [43]. In the past, other certifiers like CoLoR/Rainbow [11] and CiME/Coccinelle [17], formalized in Coq [8], were used as well.

3.1 Termination of Rewriting

There are many different flavors of term rewriting and strategies for applying rewrite rules. Many of those have their own categories.

For standard term rewrite systems, there are categories for plain rewriting (TRS Standard ☑), relative rewriting (TRS Relative ☑), rewriting modulo equational theories (TRS Equational ☑), conditional term rewriting (TRS Conditional), context-sensitive rewriting (TRS Context Sensitive), innermost rewriting (TRS Innermost ☑), and outermost rewriting (TRS Outermost ☑). There is also a category for higher-order rewriting systems (HRS (union beta)).

Concerning string rewrite systems, there are categories for plain rewriting (SRS Standard ☑), relative rewriting (SRS Relative ☑), and cycle rewriting (Cycle Rewriting).

3.2   Termination of Programs

Regarding programming languages and related formalisms, there are categories for C programs (C), C programs restricted to integers (C Integer), Java Bytecode (Java Bytecode), Prolog programs (Prolog), Haskell programs (Haskell), integer transition systems (Integer Transition Systems), and innermost rewriting with integer term rewrite systems (Integer TRS Innermost). Concerning termination of C programs, there is an "overlap" with the *SV-COMP* competition,[1] where however the focus of the two competitions is different, since *SV-COMP* considers all kinds of verification tasks for C programs, whereas termCOMP considers termination of all kinds of programming languages. Usually, *SV-COMP* runs in winter and termCOMP runs in summer, such that in each of the competitions the new current state-of-the-art of C termination analysis is represented.

3.3   Complexity Analysis

With respect to complexity analysis, there are categories for integer transition systems (Complexity: ITS), C programs restricted to integers (Complexity: C Integer), runtime complexity of term rewrite systems (Runtime Complexity: TRS ☑), runtime complexity of innermost rewriting (Runtime Complexity: TRS Innermost ☑), and derivational complexity of term rewrite systems (Derivational Complexity: TRS ☑).

4   Tools and Techniques

In this section, we give an overview on the tools that participated in the last edition, termCOMP 2018, of the competition and highlight the main techniques used by these tools.

4.1   Termination of Rewriting

In 2018, eight tools participated in categories devoted to term rewriting. On the one hand, some tools are specifically designed for certain variants of rewriting (e.g., MultumNonMulta only handles string rewrite systems, whereas Wanda, SOL, and SizeChangeTool are mainly designed for higher-order rewriting). On the other hand, the tools AProVE, T_TT_2, NaTT, and MU-TERM participated in categories for many different variants of term rewrite systems. To prove termination of TRSs, the tools use both classical reduction orderings as well as more recent powerful improvements like dependency pairs [3], matrix interpretations [20], match-bounds [26], etc. To generate the required orderings automatically, the tools typically apply existing SAT and SMT solvers.

More precisely, AProVE [27] and T_TT_2 [39] implement the dependency pair framework [28,30] which performs termination proofs in a modular way and allows the tool to apply different termination techniques for each sub-proof.

---

[1] See https://sv-comp.sosy-lab.org.

NaTT [51] combines the dependency pair framework with the weighted path order [52]. MU-TERM [1] is particularly suitable for TRSs with modified reduction relations (like innermost, context-sensitive, equational, or conditional rewriting). The goal of the tool MultumNonMulta [31] is to demonstrate the power of a few selected methods based on matrix interpretations for termination analysis of string rewrite systems. WANDA [35] implements higher-order termination techniques based on dependency pairs [38] and higher-order orderings [32], and applies an external first-order termination tool (AProVE) as a back-end [25]. The tool SOL [29] uses an extended notion of reducibility [9] for termination proofs of rules derived from second-order algebraic theories. Finally, SizeChangeTool [10] extends the size-change termination principle [41] to higher-order rewriting.

4.2   Termination of Programs

In 2018, two tools (AProVE and UltimateAutomizer) participated in the category for termination of full C programs (which may include low-level memory operations). For C programs that only operate on integers, in addition to the two tools above, the tool VeryMax participated as well. The categories for termination of other programming languages (Java, Haskell, and Prolog) were only run as a demonstration, since in that year, only the tool AProVE analyzed their termination.

For all of these programming languages, AProVE uses an approach to transform the original program into a simple back-end language (an integer transition system or a combination of ITSs and TRSs) and to prove termination of the resulting back-end system instead [47]. In contrast, the tool UltimateAutomizer [16] uses a generalization of program paths to Büchi Automata in order to remove terminating paths. VeryMax [12] is based on a framework which allows to combine conditional termination proofs obtained using Max-SMT solvers in order to generate an (unconditional) termination proof of the program.

Termination of ITSs was analyzed by the tools VeryMax, iRankFinder, and Ctrl. Moreover, Ctrl and AProVE also analyzed termination of systems that combine ITSs and TRSs. Here, iRankFinder [19] generates lexicographic combinations of ranking functions and ranks transitions incrementally [7]. Ctrl [37] and AProVE prove termination of TRSs extended by built-in integers by suitable adaptions of termination techniques for ordinary TRSs [24, 36].

4.3   Complexity Analysis

Complexity of ITSs and of C programs on integers was analyzed by CoFloCo and AProVE, where AProVE applies two integrated sub-tools KoAT and LoAT to infer upper and lower runtime bounds for integer programs, respectively. The tool CoFloCo [21] uses a modeling with cost relations to infer amortized cost bounds, whereas KoAT [14] infers both upper runtime and size bounds for parts of the program in an alternating way. Lower bounds on the worst-case runtime are inferred by LoAT [23] by simplifying programs using an adaption of ranking

functions for lower bounds, and by a subsequent inference of asymptotic lower bounds for the resulting simplified programs.

Runtime complexity of TRSs was analyzed by AProVE and T$_C$T . While runtime complexity only considers evaluations that start with basic terms (where "algorithms" are applied to "data objects"), T$_C$T also analyzed derivational complexity of arbitrary evaluations in corresponding demonstration categories. For complexity analysis, both AProVE and T$_C$T [5] use techniques which originate from termination analysis of TRSs and which are adapted in order to infer upper bounds on the number of evaluation steps [4,44]. Moreover, the tools also infer lower bounds on the (worst-case) runtime using an extension of the concept of *loops* in order to detect rules that are guaranteed to result in certain asymptotic lower bounds [22].

5   Conclusion

In this short paper, we gave a brief summary of the termination and complexity competition (termCOMP), described its organization and its different categories, and presented an overview on recent tools that participated in the competition. The competition is always open to introduce new categories in order to reflect the continuing development in the area. It also welcomes the submission of new termination and complexity problems, especially problems that come from applications. Thus, it strives to remain the main competition in the field of automated termination and complexity analysis.

References

1. Alarcón, B., Gutiérrez, R., Lucas, S., Navarro-Marset, R.: Proving termination properties with MU-TERM. In: Johnson, M., Pavlovic, D. (eds.) AMAST 2010. LNCS, vol. 6486, pp. 201–208. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-17796-5_12

2. Albert, E., Gordillo, P., Rubio, A., Sergey, I.: GASTAP: a gas analyzer for smart contracts. CoRR abs/1811.10403 (2018). https://arxiv.org/abs/1811.10403

3. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoret. Comput. Sci. **236**(1–2), 133–178 (2000). https://doi.org/10.1016/S0304-3975(99)00207-8

4. Avanzini, M., Moser, G.: A combination framework for complexity. Inf. Comput. **248**, 22–55 (2016). https://doi.org/10.1016/j.ic.2015.12.007

5. Avanzini, M., Moser, G., Schaper, M.: T$_C$T Tyrolean complexity tool. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 407–423. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_24

6. Avanzini, M., Sternagel, C., Thiemann, R.: Certification of complexity proofs using CeTA. In: Fernández, M. (ed.) RTA 2015, LIPIcs, vol. 36, pp. 23–39 (2015). https://doi.org/10.4230/LIPIcs.RTA.2015.23

7. Ben-Amram, A.M., Genaim, S.: On multiphase-linear ranking functions. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 601–620. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_32

8. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development-Coq'Art: The Calculus of Inductive Constructions. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-662-07964-5

9. Blanqui, F.: Termination of rewrite relations on λ-terms based on Girard's notion of reducibility. Theoret. Comput. Sci. **611**, 50–86 (2016). https://doi.org/10.1016/j.tcs.2015.07.045

10. Blanqui, F., Genestier, G.: Termination of $\lambda\Pi$ modulo rewriting using the size-change principle. In: Lucas, S. (ed.) WST 2018, pp. 10–14 (2018). http://wst2018.webs.upv.es/wst2018proceedings.pdf

11. Blanqui, F., Koprowski, A.: CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. Math. Struct. Comput. Sci. **21**(4), 827–859 (2011). https://doi.org/10.1017/S0960129511000120

12. Borralleras, C., Brockschmidt, M., Larraz, D., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving termination through conditional termination. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 99–117. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_6

13. Boyer, R.S., Moore, J.S.: Proving theorems about LISP functions. J. ACM **22**(1), 129–144 (1975). https://doi.org/10.1145/321864.321875

14. Brockschmidt, M., Emmes, F., Falke, S., Fuhs, C., Giesl, J.: Analyzing runtime and size complexity of integer programs. ACM Trans. Program. Lang. Syst. **38**(4), 13:1–13:50 (2016). http://dl.acm.org/citation.cfm?id=2866575

15. Buchberger, B.: Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems. Aequationes Math. **4**, 373–383 (1970). http://eudml.org/doc/136098

16. Chen, Y., et al.: Advanced automata-based algorithms for program termination checking. In: Foster, J.S., Grossman, D. (eds.) PLDI 2018, pp. 135–150 (2018). https://doi.org/10.1145/3192366.3192405

17. Contejean, E., Courtieu, P., Forest, J., Pons, O., Urbain, X.: Automated certified proofs with CiME3. In: Schmidt-Schauß, M. (ed.) RTA 2011, LIPIcs, vol. 10, pp. 21–30 (2011). https://doi.org/10.4230/LIPIcs.RTA.2011.21

18. Dehn, M.: Über unendliche diskontinuierliche Gruppen. Math. Ann. **71**(1), 116–144 (1911)

19. Doménech, J.J., Genaim, S.: iRankFinder. In: Lucas, S. (ed.) WST 2018, p. 83 (2018). http://wst2018.webs.upv.es/wst2018proceedings.pdf

20. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of term rewriting. J. Autom. Reasoning **40**(2–3), 195–220 (2008). https://doi.org/10.1007/s10817-007-9087-9

21. Flores-Montoya, A.: Upper and lower amortized cost bounds of programs expressed as cost relations. In: Fitzgerald, J., Heitmeyer, C., Gnesi, S., Philippou, A. (eds.) FM 2016. LNCS, vol. 9995, pp. 254–273. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48989-6_16

22. Frohn, F., Giesl, J., Hensel, J., Aschermann, C., Ströder, T.: Lower bounds for runtime complexity of term rewriting. J. Autom. Reasoning **59**(1), 121–163 (2017). https://doi.org/10.1007/s10817-016-9397-x

23. Frohn, F., Naaf, M., Hensel, J., Brockschmidt, M., Giesl, J.: Lower runtime bounds for integer programs. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNCS (LNAI), vol. 9706, pp. 550–567. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40229-1_37

24. Fuhs, C., Giesl, J., Plücker, M., Schneider-Kamp, P., Falke, S.: Proving termination of integer term rewriting. In: Treinen, R. (ed.) RTA 2009. LNCS, vol. 5595, pp. 32–47. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02348-4_3

25. Fuhs, C., Kop, C.: Harnessing first order termination provers using higher order dependency pairs. In: Tinelli, C., Sofronie-Stokkermans, V. (eds.) FroCoS 2011. LNCS (LNAI), vol. 6989, pp. 147–162. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24364-6_11

26. Geser, A., Hofbauer, D., Waldmann, J.: Match-bounded string rewriting systems. Appl. Algebra Eng. Commun. Comput. **15**(3–4), 149–171 (2004). https://doi.org/10.1007/s00200-004-0162-8

27. Giesl, J., et al.: Analyzing program termination and complexity automatically with AProVE. J. Autom. Reasoning **58**(1), 3–31 (2017). https://doi.org/10.1007/s10817-016-9388-y

28. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. J. Autom. Reasoning **37**(3), 155–203 (2006). https://doi.org/10.1007/s10817-006-9057-7

29. Hamana, M.: How to prove your calculus is decidable: practical applications of second-order algebraic theories and computation. Proc. ACM Program. Lang. **1**(ICFP), 22:1–22:28 (2017). https://doi.org/10.1145/3110266

30. Hirokawa, N., Middeldorp, A.: Automating the dependency pair method. Inf. Comput. **199**(1–2), 172–199 (2005). https://doi.org/10.1016/j.ic.2004.10.004

31. Hofbauer, D.: MultumNonMulta at TermComp 2018. In: Lucas, S. (ed.) WST 2018, p. 80 (2018). http://wst2018.webs.upv.es/wst2018proceedings.pdf

32. Jouannaud, J., Rubio, A.: Polymorphic higher-order recursive path orderings. J. ACM **54**(1), 2:1–2:48 (2007). https://doi.org/10.1145/1206035.1206037

33. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. **3**(2), 147–195 (1969)

34. Knuth, D.E., Bendix, P.: Simple word problems in universal algebras. In: Leech, J. (ed.) Computational Problems in Abstract Algebra, pp. 263–297 (1970)

35. Kop, C.: Higher order termination. Ph.D. thesis, VU University Amsterdam (2012). https://www.cs.ru.nl/~cynthiakop/phdthesis.pdf

36. Kop, C., Nishida, N.: Term rewriting with logical constraints. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) FroCoS 2013. LNCS (LNAI), vol. 8152, pp. 343–358. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40885-4_24

37. Kop, C., Nishida, N.: Constrained term rewriting tooL. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) LPAR 2015. LNCS, vol. 9450, pp. 549–557. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48899-7_38

38. Kop, C., van Raamsdonk, F.: Dynamic dependency pairs for algebraic functional systems. Logical Methods Comput. Sci. **8**(2) (2012). https://doi.org/10.2168/LMCS-8(2:10)2012

39. Korp, M., Sternagel, C., Zankl, H., Middeldorp, A.: Tyrolean termination tool 2. In: Treinen, R. (ed.) RTA 2009. LNCS, vol. 5595, pp. 295–304. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02348-4_21

40. von der Krone, S., Muhl, R., Waldmann, J.: star-exec-presenter (Software) (2014). https://github.com/jwaldmann/star-exec-presenter/

41. Lee, C.S., Jones, N.D., Ben-Amram, A.M.: The size-change principle for program termination. In: Hankin, C., Schmidt, D. (eds.) POPL 2001, pp. 81–92 (2001). https://doi.org/10.1145/360204.360210

42. McCarthy, J.: Recursive functions of symbolic expressions and their computation by machine, part I. Commun. ACM **3**(4), 184–195 (1960). https://doi.org/10.1145/367177.367199

43. Nipkow, T., Wenzel, M., Paulson, L.C. (eds.): Isabelle/HOL - A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45949-9

44. Noschinski, L., Emmes, F., Giesl, J.: Analyzing innermost runtime complexity of term rewriting by dependency pairs. J. Autom. Reasoning **51**(1), 27–56 (2013). https://doi.org/10.1007/s10817-013-9277-6

45. Space/Time Analysis for Cybersecurity (STAC). https://www.darpa.mil/program/space-time-analysis-for-cybersecurity

46. Sternagel, C., Thiemann, R.: The certification problem format. In: Benzmüller, C., Woltzenlogel Paleo, B. (eds.) UITP 2014. EPTCS, vol. 167, pp. 61–72 (2014). https://doi.org/10.4204/EPTCS.167.8

47. Ströder, T., et al.: Automatically proving termination and memory safety for programs with pointer arithmetic. J. Autom. Reasoning **58**(1), 33–65 (2017). https://doi.org/10.1007/s10817-016-9389-x

48. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: a cross-community infrastructure for logic solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS (LNAI), vol. 8562, pp. 367–373. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08587-6_28

49. Thiemann, R., Sternagel, C.: Certification of termination proofs using CeTA. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 452–468. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03359-9_31

50. Yamada, A.: starexec-master (Software) (2018). https://github.com/AkihisaYamada/starexec-master

51. Yamada, A., Kusakari, K., Sakabe, T.: Nagoya termination tool. In: Dowek, G. (ed.) RTA 2014. LNCS, vol. 8560, pp. 466–475. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08918-8_32

52. Yamada, A., Kusakari, K., Sakabe, T.: A unified ordering for termination proving. Sci. Comput. Program. **111**, 110–134 (2015). https://doi.org/10.1016/j.scico.2014.07.009