Report on the Termination Competition 2008

Johannes Waldmann (editor)
Fakultät Informatik, Mathematik und Naturwissenschaften,
Hochschule für Technik, Wirtschaft und Kultur, Leipzig, Germany
`http://www.imn.htwk-leipzig.de/~waldmann/`

Abstract

The Termination Competition is a yearly event where software for proving termination automatically is applied to a variety of termination problems. The fifth Termination Competition was hosted by the Computational Logic group at the University of Innsbruck, Austria, and took place in November 2008. We present the results of this competition, and some conclusions.

1 Introduction

The *Termination Competition* is an annual event to showcase and compare software that automatically proves termination of rewriting systems and logic and functional programs.

The goal of the competition is to encourage research for new termination techniques, as well as efficient implementation and combination of known techniques; and at the same time to show outside the community that automated termination provers are mature, reliable and ready to be used in applications.

The precursor of the current termination competition was an exhibition of termination provers, organized by Albert Rubio, during the Workshop on Termination 2003 in Valencia. There, the participants decided to make this a regular event, using specific sets of termination problems, a specific procedure for submitting provers and running them on an execution platform, as well as a unified presentation of results on the web.

Past termination competitions were run by Claude Marché and Albert Rubio in 2004 [1], and by Claude Marché and Hans Zantema in 2005, 2006, and 2007 [3, 4, 2]. Since 2008, the competition is hosted by the Computational Logic Group at the University of Innsbruck. The host provides a hardware and software platform for the automated submission and execution of termination provers. See Section 2 for technical detail.

A *Termination problem data base* (TPDB) was established in 2003, based on termination problems collected from publications in the field, and growing continually by submissions of new termination problems, mostly by authors of termination provers, but also from applications areas. E.g., TPDB contains most of the functions from the Haskell Prelude. In Section 3 we give some detail on the semantics and current contents of TPDB.

The termination competition of 2008 was executed in three stages at the end of the year (and beginning of the next year), and this paper gives an account of the participants (in Section 4) and results (following sections). In particular we discuss some recent developments: 2008 was the first competition that included automated investigations of derivational complexity (see Section 9), and the second competition with (optional) automated verification of termination proofs (see Section 10). We conclude with some comments on the future of the competition in Section 11.

Strategic decisions for the competition and its future are made by a steering committee, consisting of representatives of the participating research groups. Developers and users of automated termination provers subscribe to the `termtools` mailing list, and there is a central information resource for the termination community `http://termination-portal.org/`, hosted by Lehr- und Forschungsgebiet Informatik 2 of RWTH Aachen.

This report is based on contributions by Simon Bailey (Platform), Frederic Blanqui (Certification), Jürgen Giesl (LP/FP/SRS/TRS), and Georg Moser (Complexity). The views expressed in Section 11 are the editor's.

2 Platform

After deciding to move the competition to Innsbruck, the Computational Logic Group in Innsbruck designed and implemented a platform for managing and running the competition. The main capabilities required were:

- Author-driven submission of termination tools
- Automated competition scheduling and running
- Web interface for viewing results

The first version of the competition platform consists of two components: the web- interface itself and a scheduler with which the web interface interacts. The competition run in November 2008 showed that the platform per se is robust and fulfils the requirements, but also has a lot of room for improvement.

2.1 Software Environment

The platform was programmed using JavaEE, the web-framework used is JBoss Seam. The complete interface runs on a dedicated JBoss server on the competition server. The scheduling component is also written in Java and uses Quartz as the job management framework. The database back-end is a PostgreSQL server.

2.2 Hardware Platform

The competition platform currently runs on a Sunfire x4600 with 8 Dual-Core AMD Opteron CPUs and 64GB of RAM. This system is a 64-bit system and offers tools the possibility of implementing and using parallel algorithms as compared to the previous versions of the competition which were limited to single CPUs.

2.3 Current Development

Currently, two new components for the execution platform are in development: a component allowing users to submit custom-built experiments on the reference competition hardware and a component which will allow statistical analysis of the results from previous competitions.

3 Termination Problems

We give an overview of the kind of termination problems that are present in the Termination Problem Data Base (TPDB), and are being used in competitions.

In the most general view, each entry in the data base specifies a relation. The task of the termination prover is to analyze this relation.

3.1 Specifying Computations

The relation describes one step of the computation in the following models:

- a rewriting system (given by a set of rewrite rules),
- a logic program (given in Prolog notation),

- a functional program (given in Haskell notation).

These relations are classified by additional criteria. For term rewriting systems, we have the syntactic criterion on the signature:

- string rewriting (all function symbols are unary),
- term rewriting (arbitrary signatures).

Then, several variants are used in defining the relation from the set(s) of rules. The rewrite relation can be restricted by *strategies*:

- no strategy (reduce any redex), • innermost, • outermost,
- context-sensitive (using replacement maps).

The relation can also be restricted by prescribing *start terms*.

- no restrictions,
- start term is constructor-based. (The resulting complexity notion is called *run time* complexity.)
- start term is a Prolog query, or a Haskell expression. (This is mandatory for the respective categories.)

The relation $\to_1$ defined by one set of rules can be combined with some other relation $\to_2$:

- no combination (just one relation $\to_1$)
- *relative* rewriting: consider $\to_1 \circ \to_2^*$,
- *equational* rewriting: consider $\to_1 \circ \leftrightarrow_2^*$.

  In particular, $\leftrightarrow_2$ may encode commutativity, associativity or idempotency of function symbols, and this is called "rewriting modulo theories".

This gives a huge design space, and not all combinations are actually present (and some would have no reasonable semantics).

3.2 Properties of Computations

For relations that are given by the above specifications, two types of questions are considered by the provers:

- *Termination*: there is no infinite derivation,
- for terminating relations, *Derivational complexity*: the maximal length of a derivation, as function of the size of the start configuration.

The answers produced by the provers (YES, NO, or complexity bounds) are to be accompanied by a proof that is given in a form

- that can be verified by a mechanical proof checker,
- or at least be believed after human inspection.

We remark that not all of the 2×2 combinations of the above features do occur at present. The ultimate goal is indeed that a prover's output is as detailed as possible (i.e., it contains complexity information), and as trustworthy as possible (i.e., mechanically verifiable) but this is certainly a long-term process.

3.3 Current Contents of TPDB

The termination competition 2008 used version 5 of the Termination Problems Data Base. Ignoring the refinements discussed above (strategies, theories, etc.) its summarized contents is:

combined categories	FP	LP	SRS	TRS
number of problems	1676	351	777	2036

.

The data base is freely available from `http://termination-portal.org/wiki/TPDB`.

4 Participants of the Competition

A termination prover could register for a subset of the categories, and was then applied to all problems from the respective categories.

We list here the provers and their categories. In some cases, specific versions (per category) were used. We ignore these in this overview, and give a detailed description in the following sections.

- AProVE (Automated Program Verification Environment)
  - developed at RWTH Aachen, Germany by Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Swiderski, Carsten Fuhs, Carsten Otto et al.
  - home page: `http://aprove.informatik.rwth-aachen.de`
  - categories: FP, LP, SRS and TRS both non-certified and certified
- CaT (Complexity and Termination)
  - developed at LFU Innsbruck, Austria, by Martin Korp, Christian Sternagel and Harald Zankl
  - home page: `http://cl-informatik.uibk.ac.at/software/cat/`
  - categories: Complexity
- Cime3 (Complet*i*on Modulo E)
  - developed at Cédric (CNAM/ENSIIE) and LRI (Univ. Paris-Sud, CNRS), France, by Évelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons, Xavier Urbain,
  - home page: `http://a3pat.ensiie.fr/pub/index.en.html`
  - categories: SRS and TRS certified
- Jambox (Java Matchbox)
  - developed at Vrije Universiteit Amsterdam, Netherlands, by Jörg Endrullis
  - home page: `http://joerg.endrullis.de/`
  - categories: SRS and TRS non-certified
- matchbox
  - developed at HTWK Leipzig, Germany, by Johannes Waldmann,
  - home page: `http://dfa.imn.htwk-leipzig.de/matchbox/`
  - categories: SRS and TRS certified
- nonloop

- developed at HTWK Leipzig, Germany, by Martin Oppelt,
- home page: `http://dfa.imn.htwk-leipzig.de/nonloop/`
- categories: SRS non-certified

- Polytool (Proving Termination Automatically based on Polynomial Interpretations)
  - developed at KU Leuven, Belgium, by Manh Thang Nguyen and Danny De Schreye, Peter Schneider-Kamp and Jürgen Giesl,
  - home page: `http://www.cs.kuleuven.be/~manh/polytool/`
  - categories: LP
- TCT (Tyrolean Complexity Tool)
  - developed at LFU Innsbruck, Austria, by Martin Avanzini, Georg Moser, Andreas Schnabl,
  - home page: `http://cl-informatik.uibk.ac.at/software/tct/`
  - categories: Complexity
- TTT2 (Tyrolean Termination Tool 2)
  - developed at LFU Innsbruck, Austria, by Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp
  - home page: `http://colo6-c703.uibk.ac.at/ttt2`
  - categories: SRS and TRS non-certified
- TrafO (Transformation of Outermost Termination)
  - developed at TU Eindhoven, The Netherlands, by Matthias Raffelsieper and Hans Zantema,
  - home page: `http://www.win.tue.nl/~mraffels/trafo.html`
  - categories: TRS with outermost strategy

In the following sections we report on the results of the competition, by category.

5 Term Rewriting

In these categories, the tools try to prove or disprove termination of term rewrite systems. Unfortunately, this year several of the well-known tools in the area did not participate. In the future, one should try to motivate their authors to join this category again. Moreover, one should also encourage the tools to join as many of the different TRS-sub-categories as possible.

5.1 Full Termination of Term Rewriting

In the category "*TRS Standard*", one regards ordinary TRSs. Here, three tools participated on 1391 examples. The winner was AProVE which could prove termination for 995 TRSs and disprove termination for 231 examples. The second best tool was TTT2 which proved termination for 792 examples and disproved it for 178 TRSs. Finally, Jambox proved termination of 750 examples and disproved it for 60 TRSs. New techniques employed by the tools in this competition include polynomial interpretations that also allow functions like "maximum" and "minimum" as well as rational polynomial interpretations (where the search for the interpretations is based on SAT-solving).

It is remarkable that $\mathsf{T_TT_2}$ was using at most 5 seconds per problem, and its total time (for YES answers) is an eighth of the winner's. The authors wanted to show that even with severely limited resources, a considerable score could be achieved.

5.2 Innermost Termination of Term Rewriting

The categories "*TRS Innermost*" and "*TRS Outermost*" regard term rewriting under the innermost resp. the outermost evaluation strategy. In the innermost category, only AProVE participated (proving innermost termination of 241 examples and disproving it for 4 examples out of 358). Here, due to new techniques developed for disproving innermost termination, it would be interesting in the future to try to also disprove innermost termination for the other TRS-examples that are known to be non-terminating for full rewriting.

5.3 Outermost Termination of Term Rewriting

In the outermost category, a competition took place a month later than the rest of the competition. Here, several new results had been developed. In this category, JamboxGoesOut was the winner for proving termination due to its new transformation from outermost to context-sensitive rewriting. It proved outermost termination for 72 out of 291 examples (but it did not disprove outermost termination for any example). Concerning disproving outermost termination, $\mathsf{T_TT_2}$ was the winner due to its new method to decide whether a loop is also an outermost loop. It disproved outermost termination for 158 examples (but it did not prove outermost termination for any example). The remaining two participants were TrafO (which proved 46 and disproved 30 examples) and AProVE (which proved 27 and disproved 37 examples). TrafO transforms outermost termination problems to standard termination problems and then calls Jambox or AProVE to solve them.

5.4 Termination Modulo Theory

Here, one proves termination modulo AC. Unfortunately, only AProVE participated and proved termination for 57 and disproved it for 2 out of 71 examples.

5.5 Relative Termination

Here, one proves relative termination. Only Jambox participated and proved termination for 24 out of 40 examples.

The low number of participants seems strange. It is safe to assume that each termination prover contains some mechanism for reducing termination problems by "removing rules". By making this mechanism explicit, one should obtain a prover for relative termination with little extra work. In turn, progress in relative termination methods and implementations would benefit all provers.

5.6 Context-Sensitive Termination of Term Rewriting

Here, a lot of new developments took place in the last years. In 2006, a dependency pair approach for context-sensitive rewriting was developed and implemented in Mu-Term. In 2008, this approach was improved considerably to a context-sensitive dependency pair *framework*. This framework is implemented in AProVE which was the only participant in this category this year. This is a pity, since both Mu-Term and Jambox also support context-sensitive rewriting (see the outermost category above). In the future, these other tools should participate in this category as well. In 2008, AProVE proved termination of 94 out of 109 examples.

6 String Rewriting

These categories are similar to the TRS categories, but here one only regards TRSs where the function symbols have arity 1. Again, it is a pity that many of the tools that have been working on string rewrite systems in the past did not participate in the competition in 2008.

The previous winner *Matchbox* did participate in the certified string rewriting category instead. Recent progress in certification has narrowed the gap between the results in these two categories, see the discussion in Section 10.

For full termination of SRSs, the tools $\mathsf{T_TT_2}$ and AProVE that used to be tailored more to general TRSs than to SRSs were the most powerful ones. It is unclear whether this is due to the fact that tools which are more tailored to SRSs did not participate in this category in 2008 or whether the TRS-tools really improved so much on SRSs.

The winner for proving termination in this category was $\mathsf{T_TT_2}$ which proved termination for 512 SRSs and disproved it for 40 followed by AProVE which proved termination for 501 and disproved it for 22 SRSs. Jambox proved termination of 252 SRSs. For disproving termination of SRSs, the new tool nonloop was the winner by disproving termination of 92 SRSs. In particular, this tool is the first to also prove non-termination of some non-looping SRSs, by enumerating patterns in (forward) closures, and checking whether they are recurrent. $\mathsf{T_TT_2}$ finds loops by encoding derivations as a Boolean constraint system.

For relative termination of SRSs, only Jambox participated and proved termination for 32 out of 42 examples.

7 Logic Programming

In this category, the tools have to prove termination of Prolog programs for given sets of queries. Although there exist many tools for proving (and even disproving) termination of logic programs, only two tools joined the competition 2008. In the future, we should try to improve this and motivate the authors of the many other tools to participate as well.

Essentially, there are two approaches to prove termination of logic programs. Either one proves termination directly on the basis of the logic program or one transforms the logic program into a term rewrite system and proves termination of the TRS afterwards. In the competition, the tool Polytool used the direct approach and the tool AProVE used the transformational approach. The main feature of Polytool is that it contains several termination techniques that originate from term rewriting, but which were adapted to logic programs. The main feature of AProVE is that it can handle arbitrary logic programs via transformation (not just “well-moded” logic programs as in earlier transformational tools). Both Polytool and AProVE use SAT-solving when searching for suitable orders (in fact, Polytool uses AProVE as a back-end for this search).

In the competition, AProVE was a little more powerful than Polytool (AProVE could prove termination of 246 examples and Polytool proved termination for 238 examples out of 349). This is most likely due to the fact that AProVE offers more termination techniques than Polytool. But of course, these termination techniques might also adapted be to logic programming and integrated into Polytool in the future.

8 Functional Programming

In this category, the goal is to prove termination of Haskell programs for a given class of *start terms*. The data base currently consists of 1676 termination problems taken from standard libraries of the Hugsinterpreter. So this category is currently the only one in the termination competition where the termination problems are indeed taken from existing libraries in existing programming languages.

To indicate the importance of putting termination analysis into practical applications, this category was included in the competition although only one tool (AProVE) participated. Thus, currently this category is more a demonstration of the state of the art in this domain and it should serve as a motivation for other tools to join this category as well.

In the competition, AProVE could show termination of 1294 problems and it could disprove termination for 19 problems.

9 Complexity

In this year's termination competition for the first time a *complexity category* was present. It is a challenging topic to automatically determine upper bounds on the complexity of rewrite systems. Here complexity is measured as the maximal length of derivations, where either no restrictions on the initial terms are present (*derivational complexity*) or only constructor based terms are considered (*runtime complexity*). Additionally one distinguishes between complexities induced by full rewriting as opposed to complexities induced by specific strategies, as for example innermost rewriting. In this year this variety was represented trough four sub-categories: *derivational complexity–full rewriting*, *derivational complexity–innermost rewriting*, *runtime complexity–full rewriting*, and *runtime complexity–innermost rewriting*.

As test-bed we used the TRSs of the categories *TRS Standard* and *TRS Innermost* where TRSs that occur in both categories where considered once. For the sub-category *derivational complexity–full rewriting*, only *non duplicating* systems were tested as the current focus of the complexity category is on polynomial derivational/runtime complexity.

As test-bed the collection of all "standard" TRSs together with all TRSs with flag "(STRATEGY INNERMOST)" were used. Here TRSs in the current TPDB which only differ by the flag are only considered once. However for the sub-category *derivational complexity–full rewriting*, only *non duplicating* systems were tested as the current focus of the complexity category is on polynomial derivational/runtime complexity. In sum this amounts to 1404 examples for the full test-bed and 446 TRSs for the restricted test-bed.

In the sub-categories for derivational complexity, CaT and $\mathsf{T_CT}$ ran a competition, which was won by CaT. In the sub-categories for runtime complexity, $\mathsf{T_CT}$ ran as a demonstration.

The most challenging part in the design of this category was the definition of the competition semantics. While it was clear from the beginning that the current focus should be on *polynomial bounds*, it turned out to be not so easy to suit the existing competition semantics suitably to complexity analysers: while for the "standard" categories it has always been understood that the power of a tool should be measured by the number of successful runs the tool manages, the situation is not so clear for complexity analysis. What is a "successful run", if the prover is supposed to measure the complexity of a given TRS?

An ad-hoc solution would have been to test for polynomial bounds and the competing tools simply answer "yes" if this bound is verified. However this somehow blurs the potential of complexity analysis and does not really represent the research in this area. Hence we decided to extend the output certificate so that an (asymptotic) complexity function is rendered as output. Thus the tool providing the more accurate output (either as a lower or as an upper bound) should be considered the winner for the specific TRS tested.

In this year's competition only upper bounds have been tested as not much is currently known on how to (automatically) test for lower bounds on the complexity. Still the output format is already equipped to handle such an analysis as well. The following grammar is used:

$$\begin{aligned} \mathsf{S} &\to \texttt{NO} \mid \texttt{MAYBE} \mid \texttt{YES(F,F)} \mid \texttt{YES(?,F)} \mid \texttt{YES(F,?)} \\ \mathsf{F} &\to \texttt{O}(1) \mid \texttt{O}(n^m) \mid \texttt{POLY}, \end{aligned}$$

where m is a natural number.

As said, we currently focus on (polynomial) upper bounds. Firstly, for each TRS the competing tools are ranked, where the tool with the lowest upper bound wins with respect to the given TRS. For each place in this ranking, points are given. Finally the obtained points are summed up and the tool that achieved the most points in total wins the the competition, see `http://termination-portal.org/wiki/Complexity` for more details. This more refined competition semantics caused some overhead in its implementation, but it has proven its functionality in the execution.

Let us come back to the actual competition between CaT (short for Complexity And Termination) and TCT (short for Tyrolean Complexity Tool). Both tools are partially built up on basic libraries of TTT2 which explains the common output format. Both tools implement direct termination methods, sometimes restricted, that induce polynomial derivational complexity; for example *match-bound techniques*, *triangular matrix interpretations*, and *root labeling* are implemented.

In addition CaT uses the *arctic matrix method* as another direct termination technique and TCT incorporates transformation techniques like *weak dependency pairs*, *weak dependency graphs*, *polynomial path orders*, and *finite semantic labeling*. Note that the mentioned transformation techniques are only used in the sub-category “runtime complexity”. The main reason why CaT won amounts (i) to a cleverer strategy that uses (triangular) matrix interpretations also for higher dimensions and (ii) the use of the arctic matrix method.

For the future it seems to be of utmost importance to increase the number of participants. Moreover fresh methods are required in all sub-categories to extend the number of TRSs that can be analysed, with respect to lower and upper bounds, respectively. Clearly at the moment the TPDB is not an ideal test-bed for complexity analysis, but we expect that this will change due to addition of new examples to the TPDB in the course of future competitions.

For the moment we suggest to keep the focus on *polynomial* complexity analysis. Within this focus there are still a vast number of challenging problems, foremost the proper incorporation of incremental termination techniques like the dependency pair framework. Currently the extension to higher complexity classes and/or certification, although envisaged, feels premature.

10 Certification

In the certified categories of the termination competition 2008, the termination proofs are expressed in a formal language and checked by machine. All participants used Coq as the underlying proof checker, and one of two libraries that contain formalized knowledge on rewriting and termination:

- CoLoR (Coq Library on Rewriting and termination)
  - developed at INRIA, France, and TU Eindhoven, Netherlands, by Frederic Blanqui and Adam Koprowski,
  - home page: `http://color.inria.fr/`
- A3PAT (Assister Automatiquement les Assistants de Preuve avec des Traces)
  - developed at Cedric, INRIA Sophia-Antipolis, LaBRI, LRI-PCRI, France, by Xavier Urbain et al.

– home page: `http://a3pat.ensiie.fr/`

Matchbox uses CoLoR, Cime3 uses A3PAT, and AProVE submitted three versions: one that use each library alone, and one running these two in parallel.

Of the total 1391 termination problems for term rewriting, AProVE/CoLoR produced 558 certified proofs, followed by AProVE/mixed with 520 and Cime3/A3PAT with 485.

Of the total 732 termination problems in string rewriting, Matchbox/CoLoR produced 466 certified proofs, followed by AProVE/CoLoR with 406 and AProVE/mixed with 371.

Let us compare the results with the certified category of the previous competition. The problems used there were a strict subset of this years'.

For term rewriting, the percentage of solved (by the winner) problems goes up from 36 % to 40 %.

This increase is mainly due to the availability of the certified arctic matrix method.

In string rewriting, certification was first applied in 2008. The percentage of uncertified solutions in 2007 is 65 %, and this is nearly equal to the percentage of certified solutions in 2008, namely 64 %.

Let us now compare the number of certified and uncertified proofs (obtained by the respective winners) in 2008. In term rewriting, the certified winner got 56 % of the number of the uncertified winner, while in string rewriting, the corresponding ratio is 93 %.

This indicates that a state-of-the-art termination prover for term rewriting needs to rely on techniques that are (not yet) formally certified "half of the time", while in string rewriting, this is necessary only in much fewer cases.

Current developments in certified termination are

- to provide alternative certification libraries and back-ends (e.g., using Isabelle),
- to improve efficiency (e.g., by having the termination certificates checked by a Haskell or ML program that is derived automatically from an Isabelle or Coq text),
- to make more methods available for the standard categories (e.g., sharper dependency graph approximations for proving termination, and certification of loops for proving non-termination),
- to extend certification to other competition categories (e.g., relative termination).

The long-term goal is to have *all* statements produced by termination provers in any category verified automatically.

We note that with the current execution platform, the actual certification of termination proofs has a serious shortcoming: the platform checks validity of a proof but not that it proves the right theorem. Indeed it happened that a prover was working on different rewriting systems than intended because of some trivial programming error in the the input parser.

It is difficult to handle such problems because different termination provers do not even agree on the formalization of the statements and theorems, since they are using different libraries and proof checkers. The Rainbow project aims to solve this problem by providing a common format for termination proofs that is independent of the prover and of the verifier. Currently, Rainbow is used as a high-level interface to the CoLoR library.

11 Discussion

We give a few observations and derive some suggestions for future development. These are necessarily biased, and need further discussion within the community.

The Termination Competition was moved successfully to the new site. The new execution platform provides better process automation, greater computing power and more flexibility.

Still, the competition faces several challenges:

- the competition needs to be more visible outside the community,
- inside the community, the competition should be made more interesting, to win new participants, and re-activate the retired ones.
- the available computing power should be utilized better.

Let us briefly discuss the technical item first.

11.1 Multi-core

For decades now, the computing power of CPUs increased steadily. Laws of nature seem to limit the speed of instruction execution on an electronic device, but the computing power is still increasing because of CPUs containing increasing numbers of computing units (cores) that operate in parallel. It is a challenge for software to match the increased hardware capabilities.

The Termination Competition community realized that and encourages participating provers to employ parallelism. The execution service for the competition runs on a 16 core machine, and the prover can use 1 minute "wall clock" time per problem. In other words, using only one of the cores (as a classical single threaded program would do) is wasting 15/16 of the available resources.

Several termination provers indeed employed parallelism, among them AProVE and Matchbox. The basic idea is that branches of the proof search tree are explored in parallel. E.g., Matchbox is looking for standard and arctic matrix interpretations for the original and the reversed string rewriting system, all at the same time. The execution of each node in the search tree involves the production of a constraint system, its treatment by an external solver, and reading back the results.

While the external solver processes will be handled by the operating system, the production of their input seems more of a challenge, as this has to be done in the prover properly. Parallelism must therefore be handled by the run time environment of the implementation language of the prover: e.g., the Java virtual machine for AProVE, and the Glasgow Haskell runtime for Matchbox.

This does not seem to scale so easily, and a particular obstacle (for Matchbox) seems to be the current "stop the world" mode of memory management (allocation and garbage collection). It is hoped that there will be progress in models for conceptually high level parallelism with efficient run-time implementations, so that a low-level "parallelizing rewrite" of termination provers can be avoided.

Another way of avoiding programming efforts is to keep a single-threaded control structure for the termination prover, but rely on constraint solvers that exploit parallelism. Constructing such solvers is an area of ongoing research in the SAT and SMT community, but the publically available constraint solvers do not seem to reflect this at the moment.

11.2 Modular and re-useable software

To increase participation in future termination competitions, new entrants need help in passing the "entry barrier".

Current participants are therefore encouraged to publish re-usable libraries for common but "boring" tasks like parsing the problem input, printing proof output, applying standard transformations, and calling standard constraint solvers. Based on that, it is easier for new participants to concentrate on implementing fresh ideas.

E.g., in the complexity categories, the new participants CaT and TCT use libraries from TTT2, so they benefitted from previous implementation efforts. Continuing in this fashion, both tools are (partly) open source under the GNU Lesser General Public License, and their source code is available from the respective web pages.

11.3 Organisation

In 2007, the community decided to move from yearly competitions to "ongoing" competitions. The goal was to have more flexibility: it should be easier to update provers and show their results.

Yet, viewing all visible executions of provers on data base problems as competitions, creates management overhead: all decisions have to be approved by the competition committee. This slowed down the process, and that is the reason why "ongoing" competitions did not really happen.

This will be improved by the planned separation of the Competition from the Execution service. This is modelled after the Lib/Exec/Comp structure in the SMT (satisfiability modulo theory) community.

Then indeed the run time of the competition could be reduced, by restricting to certain benchmarks of problems. The competition then could be run during some conference, with results being announced at the end, increasing visibility. The ranking of participants (as applied in the Complexity category) could add more excitement. Runs over the full data base could still be done at different times.

To implement such changes, the competition steering committee needs to act much more efficiently than it does presently.

References

[1] Claude Marche and Albert Rubio. The rise of the tools. `http://www.lri.fr/~marche/termination-competition/2004/`, 2004.

[2] Claude Marche, Johannes Waldmann, and Hans Zantema. The termination competition 2007. `http://www.imn.htwk-leipzig.de/~waldmann/talk/07/wst/competition/`, 2007.

[3] Claude Marche and Hans Zantema. Termination competition 2005. `http://www.lri.fr/~marche/termination-competition/2005/TC.ppt`, 2005.

[4] Claude Marché and Hans Zantema. The termination competition. In Franz Baader, editor, *RTA*, volume 4533 of *Lecture Notes in Computer Science*, pages 303–313. Springer, 2007. This is the report on the 2006 competition. `http://www.lri.fr/~marche/termination-competition/2006/reportCompetition2006.pdf`.