A Report on the Certification Problem Format*

René Thiemann

Institute of Computer Science, University of Innsbruck, Austria
rene.thiemann@uibk.ac.at

1 Introduction

Tools that perform automated deductions are available in several areas. There are SAT solver, SMT solver, automated theorem provers for first-order logic (FTP), termination analyzer, complexity analyzer, etc. In most areas, the community was able to agree on a *common* input format, like the DIMACS-, SMT-LIBv2-, TPTP-, or TPDB-format. Such a format is beneficial for several reasons. For example, users can easily try several tools on a problem, and it is possible to compare tools by running experiments on large databases of problems.

One problem when using tools for automated deduction is that they are complex pieces of software, which may contain bugs. These bugs may be harmless or they can lead to wrong answers. To this end, certification of the generated answers becomes an important task.

Of course, to certify an answer, the result of an automated deduction tool must not be just a simple yes/no-answer, but it must provide an accompanying sufficiently detailed proof which validates the answer. For satisfiability proofs and nontermination proofs, this is often simple by given the satisfying assignment or a looping derivation; however, it can become more complex for FTP where a model may be infinite, and it may also be hard to represent complex nonterminating derivations in a finite way. In contrast, for proofs of unsatisfiability or termination, often proof are compositional and consist of several basic proof steps.

We discuss some differences of these compositional proofs in order to illustrate the special demands that arise for a format for termination proofs of term rewrite systems (TRSs).

- *Complexity of basic proof steps*: A proof of unsatisfiability for SAT can be performed in various frameworks (natural deduction, resolution, DPLL), which all have very simple inference rules. Also for FTP, the basic proof steps are rather easy (natural deduction, resolution, superposition, basic step in completion procedure). In contrast, basic proof steps in SMT solvers can be complex (apply decision procedures for supported theories) and also for termination proofs of TRSs a single proof step can be complex. For example, for removing rules via matrix interpretations [6] one has to compute matrix multiplications; and for a single application of the dependency graph processor [1], one has to approximate an undecidable problem.
- *Number of basic proof steps in a compositional proof*: In comparison to SAT, SMT, and FTP, the number of proof steps in a termination proof is rather low. Consequently, termination proofs are usually small.
- *Frameworks*: In some frameworks the inference rules are mostly fixed (e.g., natural deduction or resolution). Nevertheless, efficiently finding proofs in these frameworks requires lots of research, e.g., by developing strategies how to apply the inference rules. In the DP framework for proving termination, the set of techniques is not at all fixed. Often, the power of termination tools is increased by the invention of new ways to prove termination, e.g., by inventing new reduction pairs, new transformations, etc.
- *Determinism of basic proof steps* Several proof steps are completely determined, like the rules of natural deduction or a resolution step. But there are also basic proof steps that

* This research is supported by the Austrian Science Fund (FWF) project P22767.

WST 2012: 12th International Workshop on Termination.
Editor: G. Moser; pp. 89–93

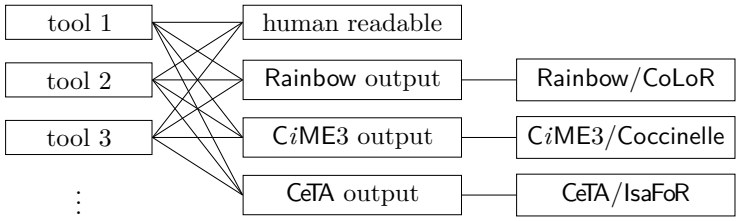

Figure 1 Certification of termination proofs before CPF

need further information to determine the result. For example, from one conflict in DPLL one can learn different conflict clauses; and for an application of the dependency graph processor, the result depends on the used approximation.

To summarize, termination proofs are usually small, but each basic proof step is complex. Moreover, the set of applied termination techniques is constantly growing.

In this report, we shortly present the certification problem format (CPF), a format developed to represent termination proofs. It has four major benefits. First, it is easy for termination tools to generate CPF files; second, it is easy to add new techniques to CPF; third, it provides enough information for certification; finally, it is a *common* proof format that is supported by several tools and certifiers.

All details on CPF and several example proofs are freely available at the following URL.

`http://cl-informatik.uibk.ac.at/software/cpf/`

2 The Certification Problem Format

Before any certifiers for termination proofs have been developed, each termination tool for TRSs provided proofs in a human-readable HTML or plain text file. From these files it was hard to extract the relevant proof steps since parameters of termination techniques are mixed with human readable explanations. Moreover, the output was not standardized at all, but every tool produced proofs in its own output format.

Hence, when the first certifiers for termination proofs have been developed (Rainbow/CoLoR [2], C*i*ME3/Coccinelle [4], and later CeTA/IsaFoR [19]), each of the certifiers demanded a proof written in their own format as input. Hence, to support certifiable proofs using all certifiers, a termination tool had to write several proof routines output, as illustrated in Figure 1.

To reduce the number of required proof outputs for termination tools, the three groups of the certifiers decided to develop *one* proof format that should be supported by all certifiers, namely CPF. Therefore, for generation of certifiable proofs, termination tools now only have to support CPF output. As CPF was also developed with several feedbacks from various termination tools it is widely accepted in the community and it is currently used as the only format during the termination competition for certified categories.

CPF is an XML-format. Choosing XML instead of ASCII was possible as termination proofs are rather small. So, the additional size-overhead of XML documents does not play such a crucial role as it might have played for (large) unsatisfiability proofs for SAT or SMT.

Using XML has several advantages: it is easy to generate, since often programming languages directly offer libraries for XML processing; even before certifiers can check the generated proofs, one can use standard XML programs to check whether a CPF file respects the required XML structure; and finally, it was easy to write a pretty printer to obtain

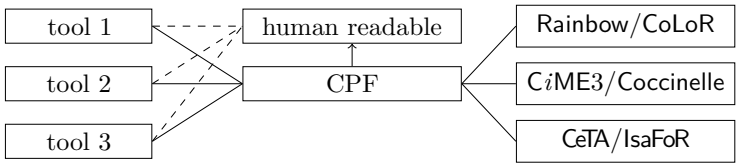

Figure 2 Certification of termination proofs using CPF

human readable proofs from CPF files. This pretty printer is written as XSL transformation (cpfHTML.xsl), so that a browser directly renders CPF proofs in a human readable way. Since this pretty printer is freely available, in principle it is no longer required for termination tool authors to write their own human readable proof output: an export to CPF completely suffices. A problem might occur if the tool uses some techniques that are not yet covered by CPF, but then it is still easily possible to extend or modify the existing pretty printer.

Note that CPF also allows to represent partial proofs: The fact, that CPF does not support all known (and in the future) developed termination techniques is reflected by allowing assumptions and by allowing intermediate results as input.

So, after the invention of CPF, the workflow and required proof export routines for certification have changed from Figure 1 to Figure 2.

3 Design decisions

In order to gain a wide acceptance for both certifiers and termination tools, representative members of the whole community have been integrated in the design process of CPF.

One major decision was that CPF should provide enough information for all three certifiers. Currently, there are some elements in CPF that are completely ignored by some certifier, which in turn are essential for another certifier.

In order to keep the burden on termination tools low, after the required amount of information has been identified, usually no further informations are required in CPF. One exception is that the proofs must be sufficiently detailed to guarantee determinism.

▶ **Example 1.** One standard technique to prove termination of a TRS $\mathcal{R}$ is to remove rules by using reduction orders [13, 10]. If the reduction order $\succ$ is provided, then usually the result is clear: it is the the remaining TRS $\mathcal{R} \setminus \succ$. So in principle, in CPF it should be sufficient to provide $\succ$. However, since there are several variants of reduction orders and since some reduction orders like polynomial orders are undecidable, it is unclear how $\succ$ is exactly defined or how it is approximated. To be more concrete, if a polynomial interpretation over the naturals is provided such that the left-hand side ℓ evaluates to $p_\ell = x^2 + 1$ and the right-hand side r to $p_r = x$, then some approximations can only detect $\ell \succsim r$ whereas a finer analysis delivers $\ell \succ r$. To avoid such problems in CPF, for rule removal it is required that the remaining system $\mathcal{R} \setminus \succ$ is also explicitly stated.

An alternative way to achieve determinism is to explicitly demand that in the proof the exact variant or approximation of the reduction pair is provided, so that the certifier can recompute the identical result. However, this alternative has the disadvantage that every variant or approximation has to be exactly specified and even worse, a certifier has to provide algorithms to compute all variants of reduction pairs that are used in termination tools. In contrast, with the current solution the certifiers can just implement one (powerful) variant / approximation of a reduction pair. Then during certification it must just be ensured that all removed rules are indeed strictly decreasing (and the remaining TRS is weakly decreasing).

Note that determinism of each proof step is also important for an early detection of errors. Otherwise, it might happen that a difference in the internal proof state of the termination tool and the state in the certifier remains undetected for several proof steps. And then errors are reported in proof steps which are perfectly okay.

▶ **Example 2.** Let $\mathcal{R} = \{\mathsf{f}(\mathsf{s}(x)) \to \mathsf{f}(\mathsf{t}(x)), \mathsf{g}(x) \to \mathsf{g}(\mathsf{s}(x))\}$. Consider a wrong proof where first the polynomial order with $\mathcal{P}ol(h(x)) = x$ for all $h \in \{\mathsf{f}, \mathsf{g}, \mathsf{s}, \mathsf{t}\}$ is used to remove the rule $\mathsf{g}(x) \to \mathsf{g}(\mathsf{s}(x))$; second, the only dependency pair $\mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{f}^\sharp(\mathsf{t}(x))$ is generated; finally, termination is proven since the dependency graph contains no edges.

If during the certification one just applies the same techniques without checking the intermediate results, then one first applies rule removal without removing any rule; second, one computes the dependency pairs including $\mathsf{g}^\sharp(x) \to \mathsf{g}^\sharp(\mathsf{s}(x))$; and finally, the error is reported that the dependency graph is not empty. Hence, the error in the first step is not detected, but in the final step—although the final step in the termination tool is sound.

Minor design decisions had to be made for all supported techniques,[1] e.g., the exact names and the exact representation of the relevant parameters, etc. For these decisions, usually the person who wanted to add a new technique to CPF was asked to provide a proposal. This proposal was then integrated into a development version of CPF and put under discussion on the CPF mailing list. Comments during the discussion were integrated in the proposal, and after the discussion has stopped, the modified proposal was then integrated into the official CPF version.

4 Problems and Future of CPF

Very recently, other classes than termination proofs were added to CPF, namely confluence proofs, completion proofs (is a TRS convergent and equivalent to an equational theory?), and equational proofs. Especially for completion proofs, the size of CPF files has grown tremendously. In experiments, example proofs of over 400 megabytes have been generated.[2] For these large proofs, both the completion tool and the certifier spend most of their time for proof export or parsing of XML documents.

Hence, action is required to counter the size-overhead of XML documents. Possibilities would include indexing of terms and rules. Moreover, for rule removal techniques, one might change CPF in such a way that the removed rules have to be provided instead of the remaining rules.[3] The latter change would also allow to represent the rule removal techniques for termination and relative termination in the same way, which in turn would allow to merge the proof techniques for termination and relative termination.

---

[1] Currently CPF supports several classes of reduction pairs (in alphabetical order): argument filters [1], matrix orders [6], polynomial orders over several carriers [13, 12, 15], recursive path orders [5], and SCNP reduction pairs [3]. Moreover, the techniques of dependency graph [1], dependency pairs [1, 8], dependency pair transformations [1, 8], loops, matchbounds [7], root labeling [16], rule removal [13, 10], semantic labeling [21], size-change termination [14, 18], string reversal, subterm criterion [10], switching to innermost termination [9], uncurrying [11, 17], and usable rules[1, 20, 8] are supported.

[2] We mention a completion proof for an example with 4 equations and 11 rules in the completed TRS. In this proof, only to show that all rules in the TRS can be derived from the equations, $\approx 90,000$ reductions have been performed, and the accumulated terms in these derivations consists of over 5 million function symbols and variables. Since symbols are strings and since there is the XML-overhead, in total, one obtains a 406 MB file (converting all symbols to integers still results in a 266 MB file). CeTA spend only 1 % of its time for checking the proof, and 99 % for parsing.

[3] If the remaining system has to specified, several steps of removing a single rule require quadratic size, whereas if one specifies the removed rules, then the size of the overall proof is linear.

However, these changes would be non-conservative changes which requires adaptations of the proof generating tools and the certifiers. Therefore, we believe that it should be discussed thoroughly by the community whether such changes should be made. Everyone is invited to contribute in the discussion.

References

1 T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.

2 F. Blanqui and A. Koprowski. CoLoR: a Coq library on well-founded rewrite relations and its application on the automated verification of termination certificates. *Mathematical Structures in Computer Science*, 21(4):827–859, 2011.

3 M. Codish, C. Fuhs, J. Giesl, and P. Schneider-Kamp. Lazy abstraction for size-change termination. In *Proc. LPAR '10*, volume 6397 of *LNCS*, pages 217–232, 2010.

4 É. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Automated certified proofs with C*i*ME3 3. In *Proc. RTA '11*, volume 10 of *LIPIcs*, pages 21–30. 2011.

5 N. Dershowitz. Termination of rewriting. *J. Symb. Comp.*, 3(1-2):69–116, 1987.

6 J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning*, 40(2-3):195–220, 2008.

7 A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On Tree Automata that Certify Termination of Left-Linear Term Rewriting Systems. *Inf. & Comp.*, 205(4):512–534, 2007.

8 J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.

9 B. Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24:3–23, 1995.

10 N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: Techniques and features. *Inf. & Comp.*, 205(4):474–511, 2007.

11 N. Hirokawa, A. Middeldorp, and H. Zankl. Uncurrying for termination. In *Proc. LPAR'08*, volume 5330 of *LNAI*, pages 667–681. 2008.

12 A. Koprowski and J. Waldmann. Arctic termination ... below zero. In *Proc. RTA'08*, volume 5117 of *LNCS*, pages 202–216, 2008.

13 D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.

14 C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *Proc. POPL '01*, pages 81–92, 2001.

15 S. Lucas. Polynomials over the reals in proofs of termination: From theory to practice. *RAIRO – Theoretical Informatics and Applications*, 39(3):547–586, 2005.

16 C. Sternagel and A. Middeldorp. Root-Labeling. In *RTA'08*, volume 5117 of *LNCS*, pages 336–350. 2008.

17 C. Sternagel and R. Thiemann. Generalized and formalized uncurrying. In *Proc. FroCoS'11*, volume 6989 of *LNAI*, pages 243–258. 2011.

18 R. Thiemann and J. Giesl. The size-change principle and dependency pairs for termination of term rewriting. *Appl. Alg. Eng. Comm. Comput.*, 16(4):229–270, 2005.

19 R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *TPHOLs'09*, volume 5674 of *LNCS*, pages 452–468. 2009.

20 X. Urbain. Automated incremental termination proofs for hierarchically defined term rewriting systems. In *Proc. IJCAR 2001*, volume LNAI 2083, pages 485–498, 2001.

21 H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.