The Certification Problem Format*

Christian Sternagel
University of Innsbruck, Austria
christian.sternagel@uibk.ac.at

René Thiemann
University of Innsbruck, Austria
rene.thiemann@uibk.ac.at

We provide an overview of CPF, the certification problem format, and explain some design decisions. Whereas CPF was originally invented to combine three different formats for termination proofs into a single one, in the meanwhile proofs for several other properties of term rewrite systems are also expressible: like confluence, complexity, and completion. As a consequence, the format is already supported by several tools and certifiers. Its acceptance is also demonstrated in international competitions: the certified tracks of both the termination and the confluence competition utilized CPF as exchange format between automated tools and trusted certifiers.

1 Introduction

Automated tools that perform logical deductions are available in several areas. For example, there are SAT solvers, SMT solvers, automated theorem provers for first-order logic (FTP), termination tools, complexity tools, confluence tools, completion tools, etc. In most areas, the community was able to agree on a *common* input format, like the DIMACS, TPTP, and TPDB formats. Such a format is beneficial for several reasons: for example, users can easily try different tools on their problems and it is possible to compare tools by running experiments on large databases of problems.

One problem when using such automated tools is that they are complex pieces of software, and thus may contain bugs. These bugs may be harmless—e.g., the tool just crashes or does not provide an answer where in principle it should be able to find one—or in the worst case lead to wrong answers. For this reason, certification of the generated answers is an important task.

Of course, to certify an answer, the result of an automated deduction tool must not be just a simple yes/no. Instead it has to provide a sufficiently detailed proof for validating the answer. Most of these proofs can be seen as a composition of several basic proof steps. But there are exceptions: for SAT and SMT, a proof of satisfiability can be given by just providing the satisfying assignment, so here no compositional proof is required.

In the following we shortly discuss some differences of the structure of these compositional proofs.

- *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, confluence or complexity proofs of term rewrite systems (TRSs) a single proof step can be complex. For example, in the termination technique called semantic labeling [33] one has to check that the given interpretation forms a model; for determining complexity of some TRS via matrix interpretations [9] one has to estimate growth rates of matrix-products, and for disproving confluence one has to ensure non-reachability under rewriting.

*This research is supported by the Austrian Science Fund (FWF) projects P22767 and J3202.

C. Benzmüller and B. Woltzenlogel Paleo (Eds.):
Eleventh Workshop on User Interfaces for Theorem Provers (UITP 2014)
EPTCS 167, 2014, pp. 61–72, doi:10.4204/EPTCS.167.8

- *Number of basic proof steps in a compositional proof*: Complexity, confluence, and termination proofs often require only a small number of proof steps in comparison to the number of steps within proofs for SAT, SMT, and FTP.
- *Set of inference rules*: The set of inference rules that are used for tools in the areas SAT, SMT, and FTP is rather static—the inference rules are fixed by the respective frameworks of natural deduction, resolution, etc. In contrast, the set of techniques that are used in confluence, complexity, and termination tools is often dynamic—much of the power of these tools relies on the invention of new ways to prove these properties, e.g., by inventing new kinds of well-founded orders, etc.
- *Determinism of basic proof steps*: Several proof steps are completely determined, like a conjunction introduction within natural deduction. But there are also basic proof steps that need further information to determine the result. For example, from one conflict in DPLL one can learn different conflict clauses.

To summarize, proofs for TRSs (with properties like termination, confluence, and complexity) are usually small in terms of number of steps, but basic proof steps may be complex. Moreover, the set of available basic proof steps is constantly growing.

In this paper, we present the certification problem format (CPF), a format initially developed to represent termination proofs for TRSs which has recently been extended to also support confluence, complexity, and completion proofs. It has four major benefits. First, it is easy for automated 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. The last point is also why the word *problem* is part of the name CPF: In the certified categories of the termination competition, first several tools produce CPF certificates, which are then used as input *problems* for different certifiers.

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

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

The main file is `cpf.xsd`, the detailed schema for CPF, which can be seen as an algebraic datatype for proofs and which formally specifies all the information that has to be provided in each proof step, in combination with clarifying comments.

2 An Example from Group Theory

Before going into a more detailed explanation of CPF, let us consider a concrete example. As already mentioned above, two of the main properties that are covered by the format are termination and confluence (of term rewrite systems). A well-known technique whose certification requires a combination of both of these properties is (Knuth-Bendix) completion [17]. That is, given a set of equations, the goal is to obtain a terminating and confluent rewrite system that serves as a decision procedure for the word problem (i.e., the question whether two terms are equal with respect to the given equations). The prime example is group theory. More precisely, given the three group axioms

$$(x \bullet y) \bullet z = x \bullet (y \bullet z) \qquad \text{(associativity)}$$
$$\mathsf{e} \bullet x = x \qquad \text{(left neutral)}$$
$$x^- \bullet x = \mathsf{e} \qquad \text{(left inverse)}$$

Completion Proof
by kbcv (version 1.7)

Input
For the following equations E

$$\bullet(\bullet(x,y),z) = \bullet(x,\bullet(y,z))$$
$$\bullet(e,x) = x$$
$$\bullet(\text{-}(x),x) = e$$

and the following TRS R

$$\bullet(\bullet(x,y),z) \to \bullet(x,\bullet(y,z))$$
$$\bullet(e,x) \to x$$
$$\bullet(\text{-}(x),x) \to e$$
$$\bullet(\text{-}(x),\bullet(x,z)) \to \bullet(e,z)$$
$$\text{-}(e) \to e$$
$$\bullet(x,e) \to x$$
$$\text{-}(\text{-}(x)) \to x$$
$$\bullet(x,\text{-}(x)) \to e$$
$$\bullet(x,\bullet(\text{-}(x),z)) \to z$$
$$\text{-}(\bullet(y,x)) \to \bullet(\text{-}(x),\text{-}(y))$$

it is proven that E is equivalent to R, and R is convergent.

Figure 1: Input specification of group theory certificate

find a set of rewrite rules, such that two given terms are equal with respect to the group axioms if and only if an exhaustive application of these rules to both of them, leads to syntactic equality. By termination of the rules an exhaustive application is always possible (since we will hit a normal form, i.e., a term to which none of the rules are applicable anymore, eventually), while confluence guarantees that we reach the same normal form independent of the employed rewriting strategy.

In the following we present a CPF certificate (as printed in a web browser, in order to increase readability) for the group theory example. As shown in Figure 1, the certificate starts by stating the kind of proof under consideration as well as the specific input (in this case a *completion proof* for the group equations). Furthermore, also the resulting TRS is part of the input. (Both are given in prefix notation, i.e., $\bullet(x,y)$ instead of $x \bullet y$ and $\text{-}(x)$ instead of x^{-}.)

The remainder of the certificate (Figure 2) contains the corresponding proof, or at least enough information that such a proof can be reconstructed by a certifier. For brevity, some parts of the proof are omitted (indicated by $\ldots$).

As stated in the certificate, for proving that the given rewrite system (R) is indeed a correct completion result, there are two obligations. First, we have to show termination and local confluence of R (the combination of which also yields confluence, by Newman's lemma). In addition, we have to make sure

Proof

1 Completion Proof

We have to prove termination and local confluence of R, and equivalence of R and E.

1.1 Rule Removal

Using the Knuth Bendix order with w0 = 1 and the following precedence and weight function … all rules could be removed.

1.2 Local Confluence Proof

All critical pairs are joinable which can be seen by computing normal forms of all critical pairs.

1.3 Equivalence Proof of R and E

R can be simulated by E as follows. All rules could be derived from the equations

$$\bullet(\bullet(x,y),z) \to \bullet(x,\bullet(y,z))$$
$$\bullet(\text{-}(x),x) \to e$$
$$\bullet(\text{-}(x),\bullet(x,z)) \leftarrow \bullet(\bullet(\text{-}(x),x),z) \to \bullet(e,z)$$
$$\ldots$$

That E can be simulated by R can be shown by just computing normal forms of each equation in E.

Figure 2: Proof part of group theory certificate

that R is equivalent to E in the sense that two terms are equal with respect to the equations in E if and only if one can be transformed into the other by applying rules of R in an arbitrary order and in arbitrary directions.

3 The Certification Problem Format

Prior to the development of certifiers for termination proofs, each termination tool provided some kind of human readable justification for its result, e.g., a plain text or HTML description of the applied techniques. It is hard to extract the relevant proof steps from this kind of justification, since parameters of termination techniques are mixed with human readable explanations. Moreover, the output was not standardized at all, i.e., every tool had its own variant.

In this setting it is not surprising that the first certifiers for termination proofs (CoLoR/Rainbow [4], Coccinelle/C*i*ME [6], and later IsaFoR/CeTA [27]) each expected their own input format. Hence, to support certifiable proofs for all certifiers, a termination tool had to provide several proof output routines, as illustrated in Figure 3.

In order to reduce the burden for termination tool authors, the development teams of the three certifiers decided to establish *a single* proof format that should be supported by all certifiers, namely CPF. As a result, nowadays termination tools only have to generate CPF certificates independent of the intended certifier. Of course, also the feedback of termination tool authors was considered during the development of CPF. By now it is widely accepted in the community and in fact the only format that is used in the certified categories of the termination competition.

Originally CPF was designed solely for termination proofs. Nowadays, it can also be used for other

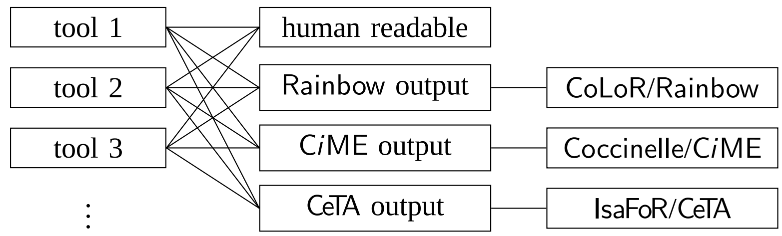

Figure 3: Certification of proofs before CPF

properties of TRSs, e.g., confluence, complexity, and completion (where it has to be checked for a given TRS $\mathscr{R}$ and equational system $\mathscr{E}$, whether $\mathscr{R}$ is a convergent TRS that is equivalent to $\mathscr{E}$). As a result, there are several tools which support CPF, namely the termination tools AProVE [12], C*i*ME [6], Matchbox [29], and $\mathsf{T_TT_2}$ [19]; the complexity tools $\mathsf{C^aT}$ [32] and $\mathsf{T_CT}$ [3]; the confluence tool CSI [31]; and the completion tools KBCV [25] and mkbTT [30].

CPF is an XML format. As an example proof, in Figure 4 we provide the internal representation of the completion proof of Section 2. Each CPF file consists of a single `<certificationProblem>` element which always has four children: the input (lines 5–10), the CPF version number (line 11), the proof (lines 12–37), and meta information (lines 38–41) which may contain tool name, configuration of the tool, source of the problem, etc.

The structure of each proof within CPF is that of an inference tree: each applied technique has to contain subproofs for its subgoals and may contain additional information. For example, the main completion proof in lines 13–36 just contains the three subproofs for local confluence (`<wcrProof>` stands for weak-Church-Rosser property, another name for local confluence), termination, and equivalence, but no further information; the termination technique of rule removal requires the additional information of the used well-founded order (line 19), the remaining TRS (line 20), and the termination proof for the remaining TRS (lines 21–23).

Choosing XML instead of a plain text format was possible as termination proofs are relatively 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. Similarly, also for other properties like confluence and complexity, the size overhead never was a problem. Only for completion proofs, we first encountered problems with too large CPF files of several hundred megabytes. However, the main problem was not the overhead due to XML, but an exponential blowup when representing a graph with sharing as fully expanded tree. Once we integrated sharing for these kinds of proofs, again the XML files became reasonably sized. As an example for this sharing, note that some of the dots (...) in Figure 2 (or equivalently, some of the `<ruleSubsumptionProof>` elements in lines 28–33 of Figure 4) represent intermediate rules which are not present in the final TRS, but which can be used to derive other conversions.

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, standard XML programs can be employed to check whether a CPF file respects the required XML structure. Finally, it was easy to write a pretty printer to obtain human readable proofs from CPF files. This pretty printer is written as an XSL transformation (`cpfHTML.xsl`), so that a browser directly renders CPF proofs in a human readable way: it just needs to be activated by adding a suitable processing instruction like line 2 of Figure 4. Since this pretty printer is freely available, in principle it is no longer required for 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

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="cpfHTML.xsl"?>
<certificationProblem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="cpf.xsd">
  <input>
    <completionInput>
      <equations> ... </equations>
      <trs> ... </trs>
    </completionInput>
  </input>
  <cpfVersion>2.1</cpfVersion>
  <proof>
    <completionProof>
      <wcrProof>
        <joinableCriticalPairsAuto/>
      </wcrProof>
      <trsTerminationProof>
        <ruleRemoval>
          <orderingConstraintProof> ... </orderingConstraintProof>
          <trs> ... </trs>
          <trsTerminationProof>
            <rIsEmpty/>
          </trsTerminationProof>
        </ruleRemoval>
      </trsTerminationProof>
      <equivalenceProof>
        <subsumptionProof>
          <ruleSubsumptionProof>
            <rule> ... </rule>
            <conversion> ... </conversion>
          </ruleSubsumptionProof>
          ...
          <ruleSubsumptionProof> ... </ruleSubsumptionProof>
        </subsumptionProof>
      </equivalenceProof>
    </completionProof>
  </proof>
  <origin>
    <proofOrigin> ... </proofOrigin>
    <inputOrigin> ... </inputOrigin>
  </origin>
</certificationProblem>
```

Figure 4: Internal representation of completion proof of Section 2

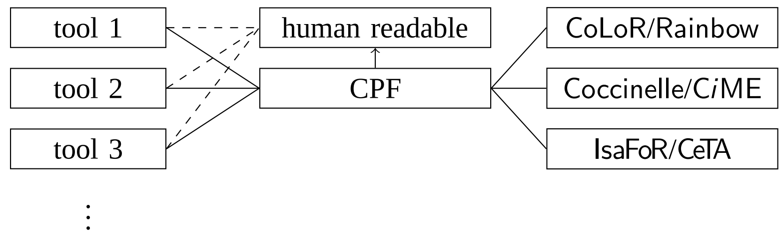

Figure 5: Certification of proofs using CPF

easily possible to extend the existing pretty printer.

So, after the invention of CPF, the workflow and required proof export routines for certification have changed from the situation depicted in Figure 3 to the less convoluted of Figure 5.

Note that CPF also allows us to represent partial proofs: The fact that CPF does not support all existing and future proof methods is reflected by allowing *assumptions*, *unknown proof steps*, and *unknown properties*.

Assumptions are useful for being able to specify and certify partial proofs. For example, even in a failed proof attempt to prove termination of some TRS $\mathscr{R}$, several steps may have been applied to simplify $\mathscr{R}$ into $\mathscr{R}'$ before no further progress was possible. With assumptions it is possible to give a CPF certificate containing all of the initial steps from $\mathscr{R}$ to $\mathscr{R}'$ and finish the proof by a termination assumption on $\mathscr{R}'$. Although such a certificate does not justify to conclude termination of $\mathscr{R}$, it can still be useful to find bugs in the tool: all steps that have been performed are checkable by a certifier. Taking this approach to the extreme, we can check individual proof steps of tools by creating certificates that specify one proof step and finish all subgoals by assumption. In this way, we added support for *online-certification* to AProVE: when enabled, then during proof search in AProVE, every single proof step is immediately exported into CPF and checked by CeTA, no matter whether the step contributes to the final proof or not. In this way some bugs in AProVE have been revealed which have not been detected for years.

Unknown proof steps are a generalization of assumptions which may be arbitrary implications like $P_1 \implies \ldots \implies P_n \implies P_0$ (for assumptions choose $n = 0$). However, unknown proof steps serve a different purpose. Whereas assumptions are there to support certification of partial proofs (e.g., when an automated tool could not derive the desired property), the purpose of unknown proof steps is to be able to specify proof steps which have not been formally defined within CPF (e.g., to specify an application of a method that has not even been published yet). To this end, each unknown proof step has to be accompanied by a short textual description. The main benefit of unknown proof steps is the ability to certify large parts of *every* generated proof of some tool, even if it uses some techniques that are unknown to the certifier.

Example 1. *Consider the following confluence proof where we assume that $\succ_1$ is some new reduction order (a well-founded order with further properties) which is not yet available in CPF.*

1. *Split the input TRS $\mathscr{R}$ into the signature disjoint systems $\mathscr{R}_1$ and $\mathscr{R}_2$. Then it suffices to prove confluence of $\mathscr{R}_1$ and $\mathscr{R}_2$ separately.*

2. *Conclude confluence of $\mathscr{R}_1$ since it is orthogonal.*

3. *Ensure confluence of $\mathscr{R}_2$ by demanding termination and local confluence.*

4. *For termination of $\mathscr{R}_2$ some rules may be removed due to $\succ_1$, resulting in $\mathscr{R}_3$.*

5. *Termination of $\mathscr{R}_3$ is concluded using a reduction order $\succ_2$.*

Whereas steps 2, 3, and 5 can be stated in CPF, currently modularity of confluence (step 1) and step 4 are not supported. However, for both steps the tool may insert unknown proof steps into the certificate. More precisely, "modularity of confluence" (confluence of $\mathscr{R}_1$ and $\mathscr{R}_2$ implies confluence of $\mathscr{R}$) for step 1 and "new reduction order $\succ_1$" (termination of $\mathscr{R}_3$ implies termination of $\mathscr{R}_2$) for step 4. Then a certifier can still check steps 2, 3, and 5 and detect potential problems in them.

Unknown properties constitute an orthogonal extension that allows us to generate certificates even for proofs relying on intermediate properties that are not yet part of CPF. The certifier can just ignore these unknown properties and check only those parts of the proof that have been specified within CPF.

We conclude this section by giving a complete list of techniques that are currently supported by CPF. There are certificates for termination and nontermination of (relative) TRSs and dependency pair problems. Regarding confluence, CPF supports certificates for local confluence, confluence, non-confluence, and completion proofs. Furthermore, CPF supports certificates for the equality and inequality of two terms with respect to a given set of equations. Another kind of certificates covers complexity (derivational and runtime) of TRSs. Finally, there is support for certificates about quasi-reductiveness of conditional rewriting.

Concerning the individual techniques, currently CPF supports several classes of reduction orders (in alphabetical order): argument filters [2], matrix orders [9], polynomial orders over several carriers [18, 20, 22], recursive path orders [7], the Knuth-Bendix order [17], and SCNP reduction orders [5]. Moreover, the techniques of dependency graph decomposition [2], dependency pairs [2,13], dependency pair transformations (instantiation, narrowing, rewriting) [2,13], loops, non-looping nontermination [8], matchbounds [11], root-labeling [23], rule removal [15,20], semantic labeling and unlabeling [33], size-change termination [21,26], string reversal, the subterm criterion [15], switching to innermost termination [14], uncurrying [16,24], and usable rules [2,13,28] are supported.

Confluence proofs are supported either directly via orthogonal or strongly closed and linear TRSs, or via Newman's lemma which requires local confluence and termination. The former by joinability of critical pairs and the latter reusing all the available machinery for termination techniques. For non-confluence CPF admits the syntactic criteria mentioned in [31], the tree-automata based techniques of [10], and the methods based on interpretations and orders of [1].

The only technique for quasi-reductiveness of conditional rewrite systems is the unraveling transformation in combination with a termination certificate.

4 Design Decisions

In order to gain a wide acceptance by certifiers as well as automated tools, representative members of the community have been integrated in the design process of CPF.

A main 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 one.

After the theoretically required amount of information has been identified, usually no further details are required in CPF (which eases the generation of certificates on the side of automated tools). One exception is that enough information must be provided to guarantee determinism of each proof step.

Example 2. *A standard technique to prove termination of a TRS $\mathscr{R}$ is to remove rules by using reduction orders [15,20]. If the reduction order $\succ$ is provided, then usually the result is clear: it is the the remaining*

TRS $\mathscr{R} \setminus \succ$. 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 others deliver $\ell \succ r$.

To avoid such problems in CPF, for rule removal and similar ambiguous techniques it is required that the certificate contains enough information to precisely compute each intermediate proof obligation. As an example, for rule removal it is sufficient to specify either the removed rules or the remaining rules explicitly, cf. line 20 in Figure 4.

An alternative way to achieve determinism is to explicitly demand that the certificate provides the exact variant or approximation of the reduction order that is employed, so that the certifier can recompute the same 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 orders that are used in termination tools. In contrast, with the current solution the certifiers can just implement one (powerful) variant / approximation of a reduction order. Then during certification it must just be ensured that all removed rules are strictly decreasing w.r.t. $\succ$ (and the remaining TRS has to be weakly decreasing w.r.t. $\succsim$).

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 an automated tool and a certifier remains undetected for several proof steps. And then errors are reported in proof steps which are perfectly okay.

Example 3. *Let $\mathscr{R}$ be a TRS consisting of the three rules A, B, and C. Now consider the following wrong proof by a termination tool:*

1. *Apply rule removal using some reduction order $\succ_1$ to remove rules A and B. (At this point assume that the termination tool contains a bug, namely that actually $\succ_1$ would only justify to remove B, but not A.)*
2. *Find some other reduction order $\succ_2$ which can remove C.*
3. *Conclude termination as there are no rules left.*

When we just apply the same techniques during certification without checking intermediate results, then we first apply rule removal with $\succ_1$ which only removes B; then we apply rule removal with $\succ_2$, which removes C; and finally, we report the error that there are rules left, namely A. This illustrates that instead of detecting the error in the first step of the tool, an error pops up in the final step of certification, although the final step in the termination tool is perfectly fine.

Minor design decisions had to be made for all the supported techniques, e.g., the exact names and the exact representation of 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.[1] Comments during the discussion were integrated in the proposal, and after the discussion has stopped, the modified proposal was integrated into the official CPF version.

[1] `cpf@informatik.uibk.ac.at`

5 Conclusion

We have presented the CPF format, an XML format that allows us to express several kinds of proofs related to term rewrite systems in a machine readable format, thereby enabling certification. So far, CPF is used by several automated tools and certifiers. New tools are of course welcome to enable CPF support as well.

Concerning future work, besides the addition of existing techniques that are currently not supported by any certifier, there is the never ending story of integrating new proof techniques into CPF. Moreover, there might be some restructuring of CPF necessary to support very large proofs. For example, indexing of terms and rules might allow us to significantly reduce the size of proofs. Furthermore, for rule removal techniques, CPF might be changed in such a way that removed rules have to be provided instead of remaining ones.[2] This change would also allow us to represent rule removal techniques for termination and relative termination in the same way, which in turn would allow us to merge proof techniques for termination and relative termination.

However, some of above changes are non-conservative and thus require adaptations of the proof generating tools and certifiers. Therefore, we believe that it should be discussed thoroughly by the community whether such changes should be made.[3] Some non-conservative changes have already been made when switching from CPF version 1.0 to version 2.0.

Another interesting point is tool collaboration, i.e., either on the producing side (e.g., several tools producing a single certificate) or on the consuming side (e.g., several certifiers taking care of different parts of a certificate). While the former is already done in practice, e.g., confluence tools rely on termination tools and reuse thusly obtained certificates when producing their own CPF output, the latter poses some trustability issues. Even if subproofs are rigorously certified by different certifiers there has to be a trustworthy way to combine those results since there is no guarantee that different certifiers are based on the same underlying semantics. We leave this crucial issue as future work.

References

[1] Takahito Aoto (2013): *Disproving Confluence of Term Rewriting Systems by Interpretation and Ordering*. In: *Proc. FroCoS*, *LNCS* 8152, Springer, pp. 311–326, doi:10.1007/978-3-642-40885-4_22.

[2] Thomas Arts & Jürgen Giesl (2000): *Termination of Term Rewriting using Dependency Pairs*. *Theoretical Computer Science* 236(1-2), pp. 133–178, doi:10.1016/S0304-3975(99)00207-8.

[3] Martin Avanzini & Georg Moser (2013): *Tyrolean Complexity Tool: Features and Usage*. In: *Proc. RTA*, *LIPIcs* 21, Dagstuhl, pp. 71–80, doi:10.4230/LIPIcs.RTA.2013.71.

[4] Frédéric Blanqui & Adam Koprowski (2011): *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), pp. 827–859, doi:10.1017/S0960129511000120.

[5] Michael Codish, Carsten Fuhs, Jürgen Giesl & Peter Schneider-Kamp (2010): *Lazy Abstraction for Size-Change Termination*. In: *Proc. LPAR*, *LNCS* 6397, Springer, pp. 217–232, doi:10.1007/978-3-642-16242-8_16.

[2] When the remaining rules are stated in the certificate, several steps of removing a single rule require quadratic size, whereas stating the removed rules, the size of the overall proof is linear.

[3] In principle it is also possible to integrate changes to some existing proof technique as a new technique which co-exists with its original obsolete version. In this way, changes can be made conservatively. But then any clean-up of outdated techniques will be a non-conservative change.

[6] Évelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons & Xavier Urbain (2011): *Automated Certified Proofs with CiME3*. In: *Proc. RTA*, *LIPIcs* 10, Dagstuhl, pp. 21–30, doi:10.4230/LIPIcs.RTA.2011.21.

[7] Nachum Dershowitz (1987): *Termination of Rewriting*. *Journal of Symbolic Computation* 3(1-2), pp. 69–116, doi:10.1016/S0747-7171(87)80022-6.

[8] Fabian Emmes, Tim Enger & Jürgen Giesl (2012): *Proving Non-Looping Non-Termination Automatically*. In: *Proc. IJCAR*, *LNCS* 7364, Springer, pp. 225–240, doi:10.1007/978-3-642-31365-3_19.

[9] Jörg Endrullis, Johannes Waldmann & Hans Zantema (2008): *Matrix Interpretations for Proving Termination of Term Rewriting*. *Journal of Automated Reasoning* 40(2-3), pp. 195–220, doi:10.1007/s10817-007-9087-9.

[10] Bertram Felgenhauer & René Thiemann (2014): *Reachability Analysis with State-Compatible Automata*. In: *Proc. LATA*, *LNCS* 8370, Springer, pp. 347–359, doi:10.1007/978-3-319-04921-2_28.

[11] Alfons Geser, Dieter Hofbauer, Johannes Waldmann & Hans Zantema (2007): *On Tree Automata that Certify Termination of Left-Linear Term Rewriting Systems*. *Information and Computation* 205(4), pp. 512–534, doi:10.1007/978-3-540-32033-3_26.

[12] Jürgen Giesl, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Steffi Swiderski & René Thiemann (2014): *Proving Termination of Programs Automatically with AProVE*. In: *Proc. IJCAR*, *LNAI* 8562, Springer, pp. 184–191, doi:10.1007/978-3-319-08587-6_13.

[13] Jürgen Giesl, René Thiemann, Peter Schneider-Kamp & Stephan Falke (2006): *Mechanizing and Improving Dependency Pairs*. *Journal of Automated Reasoning* 37(3), pp. 155–203, doi:10.1007/s10817-006-9057-7.

[14] Bernhard Gramlich (1995): *Abstract Relations between Restricted Termination and Confluence Properties of Rewrite Systems*. *Fundamenta Informaticae* 24(1-2), pp. 3–23, doi:10.3233/FI-1995-24121.

[15] Nao Hirokawa & Aart Middeldorp (2007): *Tyrolean Termination Tool: Techniques and Features*. *Information and Computation* 205(4), pp. 474–511, doi:10.1016/j.ic.2006.08.010.

[16] Nao Hirokawa, Aart Middeldorp & Harald Zankl (2013): *Uncurrying for Termination and Complexity*. *Journal of Automated Reasoning* 50(3), pp. 279–315, doi:10.1007/s10817-012-9248-3.

[17] Donald E. Knuth & Peter B. Bendix (1970): *Simple Word Problems in Universal Algebras*. In: *Computational Problems in Abstract Algebra*, pp. 263–297, doi:10.1016/B978-0-08-012975-4.50028-X.

[18] Adam Koprowski & Johannes Waldmann (2008): *Arctic Termination ... Below Zero*. In: *Proc. RTA*, *LNCS* 5117, Springer, pp. 202–216, doi:10.1007/978-3-540-70590-1_14.

[19] Martin Korp, Christian Sternagel, Harald Zankl & Aart Middeldorp (2009): *Tyrolean Termination Tool 2*. In: *Proc. RTA*, *LNCS* 5595, Springer, pp. 295–304, doi:10.1007/978-3-642-02348-4_21.

[20] Dallas Lankford (1979): *On Proving Term Rewriting Systems are Noetherian*. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA.

[21] Chin Soon Lee, Neil D. Jones & Amir M. Ben-Amram (2001): *The Size-Change Principle for Program Termination*. In: *Proc. POPL*, ACM New York, NY, USA, pp. 81–92, doi:10.1145/373243.360210.

[22] Salvador Lucas (2005): *Polynomials over the reals in proofs of termination: From theory to practice*. *RAIRO – Theoretical Informatics and Applications* 39(3), pp. 547–586, doi:10.1051/ita:2005029.

[23] Christian Sternagel & Aart Middeldorp (2008): *Root-Labeling*. In: *Proc. RTA*, *LNCS* 5117, Springer, pp. 336–350, doi:10.1007/978-3-540-70590-1_23.

[24] Christian Sternagel & René Thiemann (2011): *Generalized and Formalized Uncurrying*. In: *Proc. FroCoS*, *LNAI* 6989, Springer, pp. 243–258, doi:10.1007/978-3-642-24364-6_17.

[25] Thomas Sternagel & Harald Zankl (2012): *KBCV – Knuth-Bendix Completion Visualizer*. In: *Proc. IJCAR*, *LNCS* 7364, Springer, pp. 530–536, doi:10.1007/978-3-642-31365-3_41.

[26] René Thiemann & Jürgen Giesl (2005): *The size-change principle and dependency pairs for termination of term rewriting*. *Applicable Algebra in Engineering, Communication and Computing* 16(4), pp. 229–270, doi:10.1007/s00200-005-0179-7.

[27] René Thiemann & Christian Sternagel (2009): *Certification of Termination Proofs using CeTA*. In: *Proc. TPHOLs*, *LNCS* 5674, Springer, pp. 452–468, doi:10.1007/978-3-642-03359-9_31.

[28] Xavier Urbain (2001): *Automated incremental termination proofs for hierarchically defined term rewriting systems*. In: *Proc. IJCAR*, *LNAI* 2083, Springer, pp. 485–498, doi:10.1007/3-540-45744-5_42.

[29] Johannes Waldmann (2004): *Matchbox: A Tool for Match-Bounded String Rewriting*. In: *Proc. RTA*, *LNCS* 3091, Springer, pp. 85–94, doi:10.1007/978-3-540-25979-4_6.

[30] Sarah Winkler, Haruhiko Sato, Aart Middeldorp & Masahito Kurihara (2013): *Multi-Completion with Termination Tools*. *Journal of Automated Reasoning* 50(3), pp. 317–354, doi:10.1007/s10817-012-9249-2.

[31] Harald Zankl, Bertram Felgenhauer & Aart Middeldorp (2011): *CSI – A Confluence Tool*. In: *Proc. CADE*, *LNAI* 6803, Springer, pp. 499–505, doi:10.1007/978-3-642-22438-6_38.

[32] Harald Zankl & Martin Korp (2014): *Modular Complexity Analysis for Term Rewriting*. *Logical Methods in Computer Science* 10(1:19), pp. 1–33, doi:10.2168/LMCS-10(1:19)2014.

[33] Hans Zantema (1995): *Termination of Term Rewriting by Semantic Labelling*. *Fundamenta Informaticae* 24(1-2), pp. 89–105, doi:10.3233/FI-1995-24124.