CeTA – A Tool for Certified Termination Analysis*

Christian Sternagel René Thiemann Sarah Winkler Harald Zankl
University of Innsbruck, Austria

1 Motivation

Since the first termination competition[1] in 2004 it is of great interest, whether a proof—that has been automatically generated by a termination tool—is indeed correct. The increasing number of termination proving techniques as well as the increasing complexity of generated proofs (e.g., combinations of several techniques, exhaustive labelings, tree automata, etc.), make certifying (i.e., checking the correctness of) such proofs more and more tedious for humans. Hence the interest in automated certification of termination proofs. This led to the general approach of using proof assistants (like Coq [2] and Isabelle [12]) for certification. At the time of this writing, we are aware of the two combinations Coccinelle/C*i*ME [4, 5] and CoLoR/Rainbow [3]. Here Coccinelle and CoLoR are Coq libraries, formalizing rewriting theory. Then C*i*ME as well as Rainbow are used to transform XML proof trees into Coq proofs which heavily rely on those libraries. Hence if you want to certify a proof you need a termination tool that produces appropriate XML output, a converter (C*i*ME or Rainbow), a local Coq installation, and the appropriate library (Coccinelle or CoLoR).

In this paper we present the latest developments for the new combination IsaFoR/CeTA [13] (version 1.03). Note that the system design has two major differences in comparison to the two existing ones. Firstly, our library IsaFoR (*Isabelle Formalization of Rewriting*) is written for the theorem prover Isabelle/HOL and not for Coq. Secondly, and more important, instead of generating for each proof tree a new proof, using an auxiliary tool, our library contains several executable check-functions. Here, *executable* means that it is possible to automatically obtain a functional program (e.g., in Haskell), using Isabelle's code generation facilities [8]. For each termination technique that we have implemented in IsaFoR, we have formally proven that whenever such a check is accepted, the termination technique is applied correctly. Hence, we do not need to create an individual Isabelle proof for each proof tree, but just call the check-function for checking the whole tree (which does nothing else but calling the separate checks for each termination technique occurring in the tree). Additionally, our functions deliver error messages that are using notions of term rewriting (in contrast to error messages from a proof assistant that are not easily understandable for the novice). Furthermore, IsaFoR contains a functional parser that accepts XML proof trees. Since even this parser is written in Isabelle, we can freely choose for which programming language we want to generate code (Isabelle currently supports Haskell, OCaml, and SML). At the moment we generate Haskell code, resulting in our certifier CeTA (*Certified Termination Analysis*). However, for a user of CeTA it will make no difference if it was compiled from Haskell sources or OCaml sources.

To certify a proof using CeTA, you just need a CeTA binary plus a termination tool that is able to print the appropriate XML proof tree. Moreover, the runtime of certification is reduced significantly. Whereas it took the other two approaches more than one hour to certify all proofs during the last certified termination competition, CeTA needs about two minutes for all examples, the average time per system being 0.14 seconds. Note that CeTA can also be used for modular certification. Each single application of a termination technique can be certified by just calling the corresponding Haskell function. Another benefit of our system is its robustness. Every proof which uses weaker techniques than those formalized in IsaFoR is accepted. For example, termination provers can use the simple graph estimation of [1], as it is subsumed by our estimation.

IsaFoR, CeTA, and all details about our experiments are available at CeTA's website.[2]

*This project is supported by FWF (Austrian Science Fund) project P18763.

[1] http://termination-portal.org/wiki/Termination_Competition

[2] http://cl-informatik.uibk.ac.at/software/ceta

2 Supported Techniques

Currently, CeTA features certifying proofs for term rewrite systems (TRSs), i.e., the initial problem is always, whether a given TRS $\mathcal{R}$ is terminating or not. Hence on the outermost level of a proof we distinguish between termination and nontermination.

Termination. There are already several techniques for certifying termination proofs. Those techniques can be categorized as follows:

1. A trivial proof (for empty $\mathcal{R}$).
2. Removing some rules $\mathcal{R}'$ from $\mathcal{R}$ such that termination of $\mathcal{R} \setminus \mathcal{R}'$ implies termination of $\mathcal{R}$ [7].
3. Switching to the dependency pair (DP) framework by applying the DP transformation, resulting in the initial DP problem $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$.

In case 2, monotone linear polynomial interpretations over the naturals are supported. For 3, the following processors are available to prove finiteness of a DP problem $(\mathcal{P}, \mathcal{R})$:

Empty $\mathcal{P}$: Emptiness of the $\mathcal{P}$-component of a DP problem implies that the problem is finite.

Dependency Graph: We support a dependency graph estimation that is based on a combination of [6] and [9] (using the function `tcap`). We call this estimation EDG***. After the estimated graph is computed, $(\mathcal{P}, \mathcal{R})$ is split into the new DP problems $(\mathcal{P}_1, \mathcal{R}), \ldots, (\mathcal{P}_n, \mathcal{R})$ (one for each strongly connected component of the estimated graph). Note that our implementation allows a termination tool to use any weaker estimation than EDG***, i.e., any estimation producing a graph which contains at least those edges that are present in EDG***.

Reduction Pair: In the abstract setting of IsaFoR, the notion of *reduction pair* has been formalized. For concrete proofs there are the following instances:

- Weakly monotone linear polynomial interpretations over the natural numbers with negative constants [10]. Those can be used to remove rules from $\mathcal{P}$. Here, only the usable rules [6]—w.r.t. the argument filter that is implicit in the reduction pair—have to be oriented.
- Strictly monotone linear polynomial interpretations over the natural numbers which can be used to remove rules from both $\mathcal{P}$ and $\mathcal{R}$ (where $\mathcal{R}$ can first be reduced to the usable rules).

Nontermination. For the time being, loops are the only certifiable way of proving nontermination. If a TRS is not well-formed (i.e., $l \in \mathcal{V}$ or $\mathcal{V}\mathsf{ar}(r) \not\subseteq \mathcal{V}\mathsf{ar}(l)$ for some rule $l \to r$) the loop is implicit. Otherwise a loop is represented by a context C, a substitution σ, and terms t_1 to t_n such that $t_1 \to \cdots \to t_n \to C[t_1\sigma]$.

3 Use it for Your Termination Prover

To use CeTA for certifying your own proofs, you need a termination tool that generates appropriate XML output plus a CeTA binary (if you want to build CeTA yourself you will still need an Isabelle installation and the IsaFoR library, as well as a Haskell compiler). Hence, the main work will be to modify the termination tool in order to generate XML. In the following we will first give a short overview of the main components that are currently part of our XML format and then show how to call CeTA.

Example 3.1. As an example consider the structure of a termination proof for $\mathcal{R}$, where first a reduction pair has been used to reduce it to the TRS $\mathcal{R}'$. Afterwards the dependency pairs of $\mathcal{R}'$ are computed, resulting in a DP problem. Then the proof proceeds by applying a dependency graph estimation.

```
<proof>
  <ruleRemoval>
    <redPair>...</redPair>
    <trs>R'</trs>
    <dpTrans>
      <dps>DP(R')</dps>
      <depGraphProc>...</depGraphProc>
    </dpTrans>
  </ruleRemoval>
</proof>
```

The XML format is structured such that on the one hand, new components (like DPs after the dependency pair transformation or the reduction pair processor) are explicitly provided by the user, and on the other hand, the user cannot change components which must not be changed (e.g., the TRS when applying the DP graph processor). The general structure of proofs is as follows:[3]

$$
\begin{array}{rcl}
\langle \mathit{proof} \rangle & \stackrel{\text{def}}{=} & \texttt{<proof>}\langle \mathit{trsProof} \rangle\texttt{</proof>} \mid \texttt{<proof>}\langle \mathit{trsDisproof} \rangle\texttt{</proof>} \\
\langle \mathit{trsProof} \rangle & \stackrel{\text{def}}{=} & \texttt{<ruleRemoval>}\langle \mathit{redPair} \rangle\langle \mathit{trs} \rangle\langle \mathit{trsProof} \rangle\texttt{</ruleRemoval>} \\
& \mid & \texttt{<dpTrans>}\langle \mathit{dps} \rangle\langle \mathit{dpProof} \rangle\texttt{</dpTrans>} \\
& \mid & \texttt{<rIsEmpty/>} \\
\langle \mathit{dpProof} \rangle & \stackrel{\text{def}}{=} & \texttt{<depGraphProc>}\langle \mathit{component} \rangle^*\texttt{</depGraphProc>} \\
& \mid & \texttt{<redPairUrProc>}\langle \mathit{redPair} \rangle\langle \mathit{dps} \rangle\langle \mathit{usableRules} \rangle\langle \mathit{dpProof} \rangle\texttt{</redPairUrProc>} \\
& \mid & \texttt{<monoRedPairUrProc>}\langle \mathit{redPair} \rangle\langle \mathit{dps} \rangle\langle \mathit{trs} \rangle\langle \mathit{usableRules} \rangle\langle \mathit{dpProof} \rangle \\
& & \quad\texttt{</monoRedPairUrProc>} \\
& \mid & \texttt{<pIsEmpty/>} \\
\langle \mathit{redPair} \rangle & \stackrel{\text{def}}{=} & \texttt{<redPair>}\langle \mathit{interpretation} \rangle\texttt{</redPair>} \\
\langle \mathit{interpretation} \rangle & \stackrel{\text{def}}{=} & \texttt{<interpretation>}\langle \mathit{type} \rangle\langle \mathit{domain} \rangle\langle \mathit{interpret} \rangle^*\texttt{</interpretation>} \\
\langle \mathit{trsDisproof} \rangle & \stackrel{\text{def}}{=} & \texttt{<loop>}\langle \mathit{substitution} \rangle\langle \mathit{context} \rangle\langle \mathit{term} \rangle^*\texttt{</loop>} \\
& \mid & \texttt{<notWellFormed/>}
\end{array}
$$

For $\langle \mathit{ruleRemoval} \rangle$, the component $\langle \mathit{trs} \rangle$ holds the rules that could only be weakly oriented by the given $\langle \mathit{redPair} \rangle$. For $\langle \mathit{dpTrans} \rangle$, the dependency pairs are provided by $\langle \mathit{dps} \rangle$. The list of $\langle \mathit{component} \rangle$s within the dependency graph processor denotes all the strongly connected components (including trivial ones consisting of a single node without a self-edge) in topological order. For $\langle \mathit{interpretation} \rangle$s we currently support as $\langle \mathit{type} \rangle$ only linear polynomials and as $\langle \mathit{domain} \rangle$ only the natural numbers. (Detailed descriptions of all the other components can be found on CeTA's website.)

CeTA is called with two arguments: the first is the problem for which a proof should be certified and the second is the corresponding proof. Hence to certify the proof `proof.xml` for the problem `problem.xml`, CeTA is called as follows:

```
$ CeTA problem.xml proof.xml
```

The problem and the proof have to be in XML. For the problem the proposed XTC format—that should soon replace the TPDB format in the termination competition—is used.

Before applying a check-function to a given proof, the internal data structure is converted to XML and compared to the input string. A proof is only accepted if both are equal modulo whitespace. In this way it is ensured that (non)termination of the right TRS is proven.

[3] http://cl-informatik.uibk.ac.at/software/ceta/xml/ceta.xsd[.pdf]

4 Results and Future Work

We ran extensive tests on the 1391 TRSs from version 5.0 of the termination problems data base to evaluate the usefulness of CeTA. As termination tool we used T$_{\text{T}}$T$_2$ [11].

All tests have been performed on a server equipped with eight dual-core AMD Opteron® 885 processors running at a CPU rate of 2.6 GHz on 64 GB of system memory and with a time limit of 60 seconds for each TRS. The results can be seen in the following table (where times are given in seconds and (proof-)sizes in kilobytes).

	YES		NO		CeTA		
	#	time (avg.)	#	time (avg.)	#	time (avg.)	size (avg.)
T$_{\text{T}}$T$_2$	572	460 (0.80)	214	147 (0.68)	786	114 (0.14)	41,145 (52.35)

The YES columns of the table denote termination proofs, whereas the NO columns denote found loops. That the numbers for YES and NO sum up to 786, shows that CeTA could certify every proof generated by T$_{\text{T}}$T$_2$.

For the future we are eager to combine our attempts for a certification XML format with other approaches (like the *termination certificates grammar* of Rainbow) to obtain a standard that can then be used by all termination tools. Further, since we are working with automatically generated code, it would be possible to combine our implementation with extracted code from other formalizations in order to obtain a more powerful certifier.

References

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

[2] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development*. Texts in Theoretical Computer Science. Springer, 2004.

[3] F. Blanqui, W. Delobel, S. Coupet-Grimal, S. Hinderer, and A. Koprowski. CoLoR, a Coq library on rewriting and termination. In *Proc. WST'06*, pages 69–73, 2006.

[4] É. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Certification of automated termination proofs. In *Proc. FroCoS'07*, volume 4720 of *Lecture Notes in Artificial Intelligence*, pages 148–162, 2007.

[5] P. Courtieu, J. Forest, and X. Urbain. Certifying a termination criterion based on graphs, without graphs. In *TPHOLs'08*, volume 5170 of *Lecture Notes in Computer Science*, pages 183–198, 2008.

[6] J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In *Proc. FroCoS'05*, volume 3717 of *Lecture Notes in Artificial Intelligence*, pages 216–231, 2005.

[7] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In *Proc. RTA'04*, volume 3091 of *Lecture Notes in Computer Science*, 2004.

[8] F. Haftmann and T. Nipkow. A code generator framework for Isabelle/HOL. Technical Report 364/07, Department of Computer Science, University of Kaiserslautern, Aug. 2007.

[9] N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1-2):172–199, 2005.

[10] N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.

[11] M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. Tyrolean Termination Tool 2. In *Proc. RTA'09*, volume 5595 of *Lecture Notes in Computer Science*, pages 295–304, 2009.

[12] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[13] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proc. TPHOLs'09*, Lecture Notes in Computer Science, 2009. To appear.