Automated Verification of Termination Certificates

Frédéric Blanqui[1] and Adam Koprowski[2]

[1] INRIA & LORIA*, Campus Scientifique
BP 239, 54506 Vandoeuvre-lès-Nancy, France
[2] Eindhoven University of Technology
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands

Abstract. Termination is an important property required for total correctness of programs/algorithms. In particular it is a well studied subject in the area of term rewriting, where a number of methods and tools for proving termination has been developed over the years. Ensuring reliability of those tools is an important and challenging issue. In this paper we present a methodology and a tool for the automated verification of the results of such automated termination provers. This is accomplished by means of termination certificates, that can be easily generated by termination provers, and then by the transformation of those certificates into full formal proofs in some proof assistant/checker. This last step requires the formalization of the termination criteria used by modern termination provers. In this paper we describe the formalization of some of these criteria in the proof assistant Coq and the application of those formalizations in the transformation of termination certificates into termination proofs verifiable by Coq.

1 Introduction

In this paper we are concerned with termination of first-order term rewriting systems (TRSs) [12]. It is an important but difficult problem. Many criteria and tools for proving termination automatically have been developed over the last years. Such tools (and the proofs they produce) are getting more and more complex. This makes ensuring their reliability a big challenge. Hence, in order to be used in proof assistants or in the certification of critical systems, their results need to be formally verified.

One way to accomplish this goal is to verify the tool itself by proving that it is correct and hence its results can be trusted. This is a very hard and time-consuming task. Moreover, every change in the tool requires to redo some of the proofs. Another approach is to certify the output of the tool, every time it is used, using some proof assistant/checker. This is simpler, does not depend on the way the tool is implemented and, indeed, can be used for certifying the results of other tools. However, this requires that the tool provides enough information to easily check its results. We opt for the latter approach.

* UMR 7503 CNRS-INPL-INRIA-Nancy2-UHP

Our first contribution is CoLoR: a rich library of termination related results, developed in the proof assistant/checker Coq [22]. First we defined some general mathematical structures and then, building on that, formalized a number of modern termination criteria. Some of them were already presented in [9, 27, 28], hence in this paper we focus on the ones that were not announced until now. We describe the CoLoR library in Section 4, after introducing general preliminaries (Section 2) and discussing the general architecture of our approach (Section 3).

The second contribution is a definition of a formal grammar for representing termination certificates. This grammar is independent both of the termination prover (used to generate a certificate) and of the proof assistant (used to verify the correctness of the certificate). We will introduce it in Section 5.

Finally we developed a simple program, Rainbow, that can translate the aforementioned termination certificates into full formal proofs that can be checked by Coq with the use of the results from CoLoR. We will present the evaluation of the certification results obtained with Rainbow in Section 7 after discussing related work in Section 6.

More information about this project, as well as the Rainbow program and the CoLoR library (both including sources), can be found at:

`http://color.loria.fr/`.

2 Preliminaries

We begin by briefly recalling a few basic notions and refer to [12] for further details on term rewriting.

Let $\mathcal{V}$ be a set of variables, and Σ be a set of function symbols disjoint from $\mathcal{V}$, each symbol $f \in \Sigma$ being equipped with a fixed arity $\mathrm{ar}(f) \geq 0$. A term is either a variable or a function symbol $f \in \Sigma$ applied to $\mathrm{ar}(f)$ terms. We denote the set of terms by $\mathcal{T}(\Sigma, \mathcal{V})$. A substitution σ is a mapping from variables to terms. Its application to a term t, written $t\sigma$, replaces every occurrence of a variable x in t by $\sigma(x)$.

A *term rewriting system* (TRS) $\mathcal{R}$ over $\mathcal{T}(\Sigma, \mathcal{V})$ is a set of pairs $(\ell, r) \in \mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma, \mathcal{V})$, for which $\ell \notin \mathcal{V}$ and all variables of r occur in ℓ. Pairs (ℓ, r) are called *rewrite rules* and are usually written as $\ell \to r$.

For a TRS $\mathcal{R}$ we define a partition of its signature Σ into *defined symbols* ($\Sigma_{\mathcal{D}}$) and *constructors* ($\Sigma_{\mathcal{C}}$): a symbol $f \in \Sigma$ is *defined* if f is the root symbol of a left hand side of a rule from $\mathcal{R}$.

Given a TRS $\mathcal{R}$, let $\to_{\mathcal{R}}$ be the smallest relation containing $\mathcal{R}$ that is stable by context ($f(t_1, \ldots, t_i, \ldots, t_n) \to_{\mathcal{R}} f(t_1, \ldots, t_i', \ldots, t_n)$ whenever $t_i \to_{\mathcal{R}} t_i'$) and substitution ($t\sigma \to_{\mathcal{R}} t'\sigma$ whenever $t \to_{\mathcal{R}} t'$). We introduce two restrictions of $\to_{\mathcal{R}}$:

- $\overset{\epsilon}{\to}_{\mathcal{R}}$ where rewriting is allowed only at the root position (top steps) and
- $\overset{>\epsilon}{\to}_{\mathcal{R}}$ where rewriting is allowed anywhere but at the root position.

For an arbitrary relation $\to$ we denote its reflexive and transitive closure by $\to^*$. A binary relation $\to$ is called *terminating* or *strongly normalizing*, $\mathsf{SN}(\to)$, if it is well-founded, i.e. there exists no infinite sequence $t_0, t_1, \ldots$ such that $t_i \rightarrow t_{i+1}$ for all $i \in \mathbb{N}$. A TRS $\mathcal{R}$ is called terminating if $\mathsf{SN}(\rightarrow_{\mathcal{R}})$ holds.

A binary relation $\rightarrow_1$ is called *terminating relative to* a binary relation $\rightarrow_2$, written as $\mathsf{SN}(\rightarrow_1 / \rightarrow_2)$, if there is no infinite sequence $t_0, t_1, \ldots$ such that

- $t_i \rightarrow_1 t_{i+1}$ for infinitely many values of i, and
- $t_i \rightarrow_2 t_{i+1}$ for all other values of i.

We use the notation $\rightarrow_1 / \rightarrow_2$ to denote $\rightarrow_2^* \cdot \rightarrow_1$; it is easy to see that $\mathsf{SN}(\rightarrow_1 / \rightarrow_2)$ coincides with well-foundedness of $\rightarrow_2^* \cdot \rightarrow_1$.

Given a quasi-order $\succeq$, we define its strict part $\succ$ as $s \succ t$ iff $s \succeq t$ and $t \not\succeq s$; and the corresponding equivalence relation $\sim$ as $s \sim t$ iff $s \succeq t$ and $s \preceq t$.

The usual lexicographic and multiset extensions of an order $\succ$ are denoted by $\succ^{\mathrm{lex}}$ and $\succ^{\mathrm{mul}}$ respectively.

3 General architecture

In this section we will try to give an overview of our approach to certification of termination. Our goal is to verify the results produced by termination provers with the use of the Coq proof assistant. This is achieved by means of certificates, that is a transcription of termination proofs in a dedicated format. There are three main ingredients to our approach, which we will discuss in more detail in the remainder of this paper:

- CoLoR (Coq library on Rewriting and Termination): a Coq library consisting of formalized termination techniques (Section 4),
- TCG (Termination Certificates Grammar): a format for termination certificates (Section 5),
- Rainbow: a simple tool transforming termination certificates to formal termination proofs, verifiable by Coq (evaluated in Section 7).

We now sketch how the certification process looks like. For a given TRS $\mathcal{R}$ some termination prover is called. If it succeeds in proving termination with the currently supported methods, it outputs a termination certificate in the TCG format. This certificate is then given to Rainbow which translates it into a Coq script containing a formal proof of the claim that $\mathcal{R}$ is terminating. To accomplish that, Rainbow uses the general theorems on termination from CoLoR. Note that, thanks to the generality of those theorems, the generated scripts are quite readable. Then Coq is executed on this script (as a proof checker) to validate that the argument provided by the termination tool is correct. The information flow is summarized in Figure 1.

4 CoLoR: a Coq Library of Results on Termination

Coq [22, 4] is a proof assistant/checker based on the Calculus of Inductive Constructions (CIC) [35]. CIC is a very rich typed λ-calculus following the proofs-as-objects principle. It includes simple, inductive, dependent and polymorphic

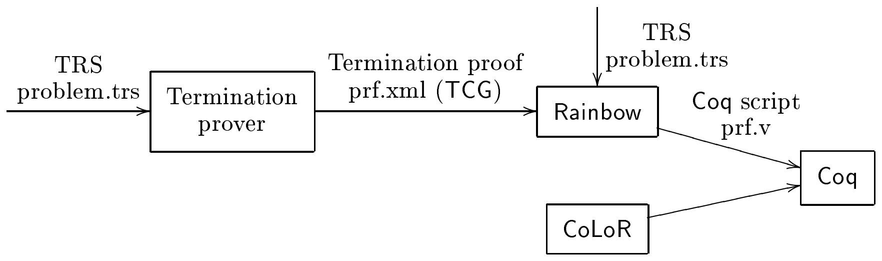

Fig. 1. Certifying termination with CoLoR and Rainbow

types (Girard' system F). It allows to define functions in an ML-like style using fixpoints and pattern matching, but recursive calls must be done on structurally smaller arguments to ensure termination. Proofs are expressed using tactics, either built-in or user-defined (Coq provides a high-level tactic language Ltac allowing to do non-linear pattern-matching on the current goals and hypotheses [10]). Built-in tactics range from basic tactics, like applying a theorem, to complex ones, like trying to automatically solve propositions in Presburger arithmetic.

In this section, we present the Coq formalization of various advanced termination criteria used in modern automated termination provers. These criteria are then used to certify termination proofs that make use of them. For doing this, there are two distinct approaches depending on the way objects (such as rewrite rules, interpretations, etc.) are formalized in the proof assistant: using a shallow or a deep embedding. In a shallow embedding, in contrast to a deep embedding, no general representation of the objects under consideration is developed in the proof assistant a priori. For instance a polynomial can be represented by a function, whereas in a deep embedding, a data type for polynomials is defined and a general theory of polynomials can be developed. Both approaches have their advantages and disadvantages. A combination of both approaches is also possible.

At the moment CoLoR uses deep embeddings only. This means that, apart from formalized termination criteria, it also contains developments of various libraries on mathematical structures, data structures and term structures that can be of general interest and may be used in other formalization efforts not necessarily related to termination and/or rewriting.

These developments have been contributed over the years by various people: Solange Coupet-Grimal, William Delobel, Léo Ducas, Sébastien Hinderer, Stéphane Le Roux, Johannes Waldmann, Hans Zantema and the authors of the present paper. Some developments were already described in details in technical reports or conference papers. We will just mention them and focus on the parts that have not been presented before.

Altogether, the CoLoR library has nearly 51000 lines of code (including comments and blank lines) with approximately 40 inductively defined predicates, 180 recursive function, 775 non-recursive definitions, 110 tactic definitions and 2700 theorems. As a comparison, the standard library of Coq8.1pl3 has 92000 lines of code, 160 inductive predicates, 315 recursive functions, 1450 definitions, 115 tactic definitions and 5335 theorems.

Since the development is huge, we can only give some overall description of it. We will however describe in more details some of the basic types and some interesting functions.

It is important to note that all CoLoR theorems are proved constructively. In the literature, proofs are generally classical, deducing a contradiction from the existence of an infinite reduction sequence. Finding a good induction argument for converting a classical proof into a constructive proof may be non-trivial (if not impossible).

4.1 Libraries on mathematical structures

We currently have two main libraries on mathematical structures: a library on relations (this includes graphs) and a library on semi-rings.

Relations and graphs A relation R on a set A (object of type Type in Coq) is represented as an object of a functional type $A \to A \to \text{Prop}$, where Prop is the type of propositions. Strong normalization of an element x of type A is defined inductively as usual: $\mathsf{SN}\ R\ x$ if, for all y such that $R\ x\ y$, $\mathsf{SN}\ R\ y$. Then, R is called terminating (or well-founded, or strongly normalizing) if every term of type A is strongly normalizing.

As already mentioned in the previous section, relative termination plays an important role in termination proofs. The termination of R relatively to E is defined as the strong normalization of the relation $E^* \cdot R$, where E^* is the reflexive and transitive closure of E.

Furthermore, various notions are defined like the notions of relation iteration, path, cycle, strongly connected component and adjacency matrix, and various basic properties are established about them.

Among the most notable things, let us mention:

- Le Roux gives constructive proofs of the following results [36]:
  - A relation over a set with decidable equality is decidable iff the transitive closures of its finite restrictions are also decidable.
  - A decidable relation is acyclic and equality on its domain is decidable iff any of its finite restrictions has a decidable linear extension.
  - A relation is acyclic and equality on its domain is decidable iff it is uniformly topologically sortable.

  Note that this work is part of a more general work on Nash equilibria.
- Ducas defined and proved correctness of a function computing the strongly connected components of a finite relation (graph), using computations on boolean adjacency matrices.
- The authors proved various general theorems about the (relative) strong normalization of the combination (union and/or composition) of various relations. For instance, $R \cdot E^*$ terminates if R terminates and $E \cdot R \subseteq R$. Or $(E \cup E')^* \cdot (R \cup R')$ terminates if both $E^* \cdot R$ and $(E \cup R)^* \cdot (E' \cup R')$ terminate.

Semi-rings A *commutative semi-ring* [19] consists of a carrier D, two designated elements $d_0, d_1 \in D$ and two binary operations $\oplus, \otimes$ on D, such that both $(D, d_0, \oplus)$ and $(D, d_1, \otimes)$ are commutative monoids and multiplication distributes over addition: $\forall x, y, z \in D : x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$.

The Coq standard library contains a notion of a commutative semi-ring corresponding to the definition presented above.[3] CoLoR builds on that but encloses the semi-ring specification within a module providing a real encapsulation and modularization. This allows for instance to define matrices over an arbitrary semi-ring of coefficients, which we will introduce in the following section. It also allows to prove a number of results following from the specification of a semi-ring that will automatically be available for any instantiation to an actual semi-ring. A few such instantiations are provided:

- standard semi-rings over natural numbers and integers (with standard addition and multiplication operations);
- arctic semi-ring: natural numbers/integers extended with $-\infty$ with max as arctic addition and with standard addition, $+$, as semi-ring multiplication;
- a tropical semi-ring: dual to arctic, *i.e.* natural numbers extended with $+\infty$ with min and $+$ operations, is work in progress.

4.2 Libraries on data structures

The CoLoR library contains many functions and theorems on basic data structures like lists, vectors, polynomials, matrices and finite multisets. We just mention some interesting points.

Lists $\mathrm{list} : \mathrm{Type} \to \mathrm{Type}$ is the polymorphic inductive type whose constructors are $\mathrm{nil} : \forall A : \mathrm{Type}, \mathrm{list}\ A$ and $\mathrm{cons} : \forall A : \mathrm{Type}, A \to \mathrm{list}\ A \to \mathrm{list}\ A$.

Blanqui gave a constructive proof of the pigeon-hole principle on lists: for all lists l_1 and l_2 such that every element of l_1 is in l_2, if the length of l_1 is strictly greater than the length of l_2, then there is an element x occurring at least twice in l_1.

Vectors $\mathrm{vector} : \mathrm{Type} \to \mathrm{nat} \to \mathrm{Type}$ is the polymorphic and dependent inductive type whose constructors are $\mathrm{Vnil} : \forall A : \mathrm{Type}, \mathrm{vector}\ A\ 0$ and $\mathrm{Vcons} : \forall A : \mathrm{Type}, A \to \forall n : \mathrm{nat}, \mathrm{vector}\ A\ n \to \mathrm{vector}\ A\ (n + 1)$. It corresponds to lists of fixed length or arrays. The use of dependent types is in general somewhat difficult (because of the type constraints generated by the dependencies) but on the other hand they are quite powerful. For instance, a term of type $\mathrm{vector}\ A\ n$ has n elements by construction. So we do not need to provide a proof of this statement. CoLoR provides a rich set of functions operating on vectors (including vector arithmetic) and results about such functions, that has been reused in a formalization of the spi-calculus [5].

[3] It is used for the inner workings of the *ring* tactic.

Polynomials Polynomials with integer coefficients and multiple indeterminates have been formalized by Hinderer [20]. In contrast to matrices or multisets, polynomials are not yet defined as a functor building a structure for polynomials given a structure for the coefficients. We however expect to change this in order to be able to certify proofs using polynomial interpretations with rational or real coefficients [32, 14].

The monomial $x_1^{k_1} \dots x_n^{k_n}$ is represented by the vector of natural numbers $(k_1, \dots, k_n)$. A polynomial $\sum_{i=1}^{p} c_i m_i$, where m_i is a monomial, is then represented by a list of pairs (c_i, m_i). A polynomial can therefore have different representations. The library provides functions to compose and decompose polynomials as well as all the basic operations on them (addition, multiplication, power, composition, evaluation on integers) and theorems on monotonicity.

Multisets Finite multisets have been formalized by Koprowski in order to prove well-foundedness of the higher-order recursive path ordering (HORPO) [24]. The main property about multisets is the fact that the multiset extension of a well-founded relation is well-founded. This, and all the other results about multisets, are proved axiomatically from a small set of functions and their specifications, using the module system of Coq. This means that all those results are independent of a particular representation of multisets. A simple implementation using lists is provided.

Matrices Matrices have been formalized by Koprowski and are used for matrix [27] and arctic [28] interpretations. Matrices are generic, *i.e.* their entries come from an arbitrary semi-ring. A number of operations on matrices is provided (matrix creation, access functions, transposition, addition and multiplication) along with proofs for a number of matrix properties (such as associativity of matrix multiplication).

4.3 Libraries on term structures

Ultimately, the CoLoR project is interested in certifying the termination of programs, for various programming paradigms: string rewrite systems, term rewrite systems, logic programs, functional programs and imperative programs. For the moment, we mainly considered the first two paradigms. Note however that logic programs and functional programs can be proved terminating by using techniques developed for rewrite systems [33, 16]. To this end, we formalized various kinds of term structures:

Strings Strings (or words) over an alphabet A are simply represented as lists of elements of type A. For the moment, few notions have been formalized on strings: context, string rewrite rule and string rewrite system reversal.

First-order varyadic terms First-order terms over a set Sig of function symbols of varyadic arity are represented by an inductive type term : Set whose constructors are Var : nat $\rightarrow$ term and Fun : Sig $\rightarrow$ list term $\rightarrow$ term. For the moment, only the notions of context, substitution and rewriting are defined.

First-order terms with fixed arity Usual first-order terms over a set Sig of function symbols of fixed arity are represented by a dependent inductive type term : Set whose constructors are Var : nat $\to$ term and Fun : $\forall f$: Sig, vector term (arity f) $\to$ term. This is the more developed term library. It contains the formalization of many notions like: interpretation (universal algebra), substitution (defined as an interpretation on terms), context, rewriting, (weak) reduction ordering/pair, syntactic unification, etc. See Example 6 in the Appendix for a development concerning this term structure and illustrating the usefulness of dependant types.

Simply-typed λ-terms Finally, CoLoR also provides a formalization of the simply-typed λ-terms, using de Bruijn representation for bound variables. This formalization was used in the proof of well-foundedness of HORPO [24]. The library is quite extensive and contains many definitions standard for the simply-typed λ-calculus along with few less standard ones (like many-variables, typed substitution or an equivalence relation on terms extending the concept of α-convertibility to free variables). Many results are provided as well, such as the subject reduction property or termination of the β-reduction. For more details we refer to [25].

4.4 Termination Results

Finally, the focus of CoLoR is on providing formalizations of actual methods for proving termination. At the moment the focus of the library is on three types of termination problems:

- full termination, *i.e.* $\mathsf{SN}(\to_{\mathcal{R}})$,
- relative termination, *i.e.* $\mathsf{SN}(\to_{\mathcal{R}} / \to_{\mathcal{S}})$ and
- relative, top termination, *i.e.* $\mathsf{SN}(\overset{\epsilon}{\to}_{\mathcal{R}} / \to_{\mathcal{S}})$, used in conjunction with the dependency pair transformation, see Theorem 1 below.

Interpretation-based methods At the moment CoLoR contains the following interpretation-based termination criteria:

- polynomial interpretations [31] (contributed by Hinderer [20]),
- matrix interpretations [13] (contributed by Koprowski and Zantema [27]),
- arctic interpretations [28] (contributed by Koprowski and Waldmann [28]).

All those methods are formalized in the setting of monotone algebras — a general framework for interpretation-based termination methods. It is incorporated in CoLoR in its full generality, making it easier to add further methods that fit into this setting.

We refer to the original papers for more details on the formalization of those termination techniques and to [27] for details on the formalization of monotone algebras.

Recursive Path Order The Recursive Path Order (RPO) [11] is an order that lifts a quasi-order on function symbols, called a *precedence*, to a quasi-order on terms. It allows for different ways of comparing function arguments: either lexicographically or using multiset order, depending on the *status* of function symbols.

The development of RPO has been contributed to CoLoR by Coupet-Grimal and Delobel [9] and we refer to the original article for details about this formalization. It contained the main result, that is well-foundedness of the order along with a number of properties. Later on some results such as decidability and stability under substitution have been added by Koprowski.

We still plan to make some adjustments to RPO before adding support for certification of proofs using this technique. Firstly, although the quasi-order variant of the ordering was formalized, its equality is just a syntactic equality instead of equivalence on terms generated by the equivalence of symbols in the precedence. Secondly the lexicographic extension currently used in RPO allows to compare only sequences of equal lengths and does not allow for permutations of arguments.

Apart from first-order RPO, its higher-order variant, the higher-order recursive path ordering (HORPO) [23], is also part of the CoLoR library. It was contributed by Koprowski [24] and contains all the essential properties of HORPO like decidability, monotonicity, stability under substitution and well-foundedness of its union with β-reduction of simply-typed lambda calculus. However higher-order rewriting itself is not formalized and would need to be before termination of higher-order rewriting systems can be certified with CoLoR. Given lack of demand and virtually no prover supporting higher-order rewriting this is unlikely to happen in the near future.

Dependency Pair Method The dependency pair method [3] is a powerful transformational method for proving termination of rewriting. It enjoys a number of extensions that all fit into the dependency pair framework [17]. The basic theorem of the dependency pair method (Theorem 1 below) has been formalized in Coq by Blanqui. Although the dependency pair framework itself is not used in the CoLoR libraries, Ducas recently made a big first step towards certification of proofs using the dependency pair approach, by formalizing Theorem 3 below.

Now we describe the basic dependency pair transformation. For every defined symbol $f \in \Sigma_{\mathcal{D}}$ we add to the signature a new marked symbol $f_{\#}$ with the same arity as f. If $f(s_1, \ldots, s_n) \to r$ is a rule in $\mathcal{R}$ and $g(t_1, \ldots, t_m)$ is a subterm of r for $g \in \Sigma_{\mathcal{D}}$, then the rewrite rule $f_{\#}(s_1, \ldots, s_n) \to g_{\#}(t_1, \ldots, t_m)$ is called a *dependency pair* of $\mathcal{R}$. The TRS consisting of all dependency pairs of $\mathcal{R}$ is denoted by $\mathsf{DP}(\mathcal{R})$.

The main result concerning the dependency pairs transformation is the following:

Theorem 1 ([3]). *Let $\mathcal{R}$ be a TRS. Then $\mathsf{SN}(\to_{\mathcal{R}})$ iff $\mathsf{SN}(\xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})} / \to_{\mathcal{R}})$.*

An important technique at the core of the dependency pair method is the analysis of the dependency graph, that is, the possible dependency pairs that can follow each other [3].

Definition 2 (Dependency graph). *The* dependency graph *of a TRS $\mathcal{R}$ is the relation $G(\mathcal{R})$ on $\mathsf{DP}(\mathcal{R})$ such that $(\ell_1 \to r_1)\ G\ (\ell_2 \to r_2)$ iff $r_1\sigma \to^*_{\mathcal{R}} \ell_2\tau$ for some substitutions σ, τ.*

The analysis of the dependency graph strongly connected components allows to split a problem into smaller sub-problems in such a way that the termination of the original problem follows from the termination of all of its sub-problems.

Theorem 3 ([15]). *Let $\mathcal{R}$ be a TRS, and $\mathrm{SCC}_1, \ldots, \mathrm{SCC}_n$ be all the strongly connected components of $G(\mathcal{R})$. Then, $\mathsf{SN}(\overset{\epsilon}{\to}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}})$ iff $\mathsf{SN}(\overset{\epsilon}{\to}_{SCC_i}/\to_{\mathcal{R}})$ for every $i \in \{1, \ldots, n\}$.*

Now, since the dependency graph is undecidable in general, termination provers use over-approximations of it. The most well-known over-approximation of the graph is based on unification: $(\ell_1 \to r_1)\ G\ (\ell_2 \to r_2)$ iff r_1' and ℓ_2 are unifiable, where r_1' is obtained from r_1 by firstly replacing all its subterms with a defined root symbol by a variable and then replacing all variables of such a term with fresh ones. For the moment, in CoLoR, we only defined the over-graph based on head-symbols equality: $(\ell_1 \to r_1)\ G\ (\ell_2 \to r_2)$ iff r_1 and ℓ_2 are headed by the same (marked) symbol. We left for future work the definition (and proof) of a function computing the most general unifier of two terms.

Argument Filtering The argument filtering method [3] is another transformational method that consists in removing some arguments of a function symbol, or replacing a function call by one of its arguments.

Definition 4 (Argument filtering [30]). *An argument filtering function is a function π such that for any $f \in \Sigma$, $\pi(f)$ is either an integer i or a list of integers $[i_1, \ldots, i_m]$ ($m \geq 0$) such that $0 \leq i, i_1, \ldots, i_m \leq \mathrm{ar}(f)$. We can naturally extend π to terms as follows:*

$$\begin{aligned} \pi(x) &= x \\ \pi(f(t_1, \ldots, t_n)) &= \pi(t_i) && \textit{if } \pi(f) = i \\ \pi(f(t_1, \ldots, t_n)) &= f(\pi(t_{i_1}), \ldots, \pi(t_{i_m})) && \textit{if } \pi(f) = [i_1, \ldots, i_m] \end{aligned}$$

And to TRSs as: $\pi(\mathcal{R}) = \{\pi(\ell) \to \pi(r) \mid \ell \to r \in \mathcal{R}\}$.

Theorem 5 ([3]). *Let $\mathcal{E}$ and $\mathcal{R}$ be TRSs over a signature Σ and π be an argument filtering function on Σ. Then, $\mathsf{SN}(\overset{\epsilon}{\to}_{\pi(\mathcal{R})}/ \to_{\pi(\mathcal{E})})$ implies $\mathsf{SN}(\overset{\epsilon}{\to}_{\mathcal{R}}/\to_{\mathcal{E}})$.*

This method, restricted to non-collapsing argument filtering functions (*i.e.* every $\pi(f)$ maps to a list of integers and not to a single integer), has been formalized in CoLoR by Blanqui. It should not be too difficult to extend the formalization in order to get rid of that restriction.

5 A Grammar for Termination Certificates

It is well known that, for a given problem, it is generally easier to check that a solution is correct than to try to find such a solution. For problems of the form "Does the object a have the property P?", where the answer is "Yes" or "No", checking the correctness of the answer requires a *certificate* c, that is some piece of data, some evidence, on which one can do some computations (simpler than checking whether a has property P) to check if the answer is correct. And even if finding a solution and checking whether a solution is correct lie in the same complexity class, the difference may be important in practice.

Our aim is to provide a high-level grammar (and its semantics) for termination certificates allowing easy transformation of such certificates to formal proofs checkable by some proof assistant. Several constraints guided us in the design of this grammar:

– The certificates should provide enough information so that they can be checked with reasonable complexity.
– The certificates should be independent of the tools used to produce them, and independent of the proof assistants used for checking them.

We developed such a grammar, TCG, for Termination Certificates Grammar, along with Rainbow — a tool to transform proofs in the TCG format into formal Coq proofs using the results from CoLoR. Due to space considerations we are unable to present the grammar in full detail here and we need to restrict its presentation to a few general remarks. However, the full TCG (as an XML schema, as the grammar is implemented using XML) with comments and examples is available via the web-page of the project, `http://color.loria.fr`, and the interested reader is encouraged to consult it.

A termination proof consists of a number of applications of well-known methods for proving termination, some of which we presented in Section 4.4. Each theorem can be presented as an inference rule with a number of premises and a conclusion. This naturally gives a tree structure to any termination proof and this tree structure is reflected in the TCG.

Each node in the tree corresponds to an instance of some theorem formalized in CoLoR. Apart from identifying the theorem to be used it contains all the information required to instantiate this theorem. For instance the node for the matrix interpretation method will contain the dimension for the matrices and a matrix interpretation for every function symbol in the signature of the problem under consideration; the node for RPO will contain an encoding of the status function and of the precedence on function symbols, etc.

The grammar is designed such that it is easy to extend it with new methods as their formalizations are added to the CoLoR library. It is also easy for the authors of the termination proving tools to use our approach to certify the results of their tools merely by adding support for TCG as another output format of their tools and making sure that the theorems they use correspond to the ones available in CoLoR. For many termination techniques this is not problematic as their theory is well-established and uniform across all termination provers.

6 Related Work

In [7], Contejean *et al* describe a Coq library called Coccinelle developed for certifying the results of the automated termination prover C*i*ME [8]. The size of Coccinelle is about half of the size of CoLoR. The approach followed in this work is slightly different from ours. Initially the C*i*ME termination prover was directly generating Coq scripts. Recently an intermediate proof format, similar to TCG, was introduced, allowing, in principle, other tools to certify their results with the use of this library. Secondly, C*i*ME uses shallow embeddings for representing rewrite relations, dependency pairs transformations and polynomial interpretations in Coq. Because of that the Coq scripts generated by C*i*ME are on average 3 times longer and less understandable than the ones generated by CoLoR, and their verification by Coq is twice longer.[4] But first and foremost we would like to stress that those two approaches are not exclusive, but rather complementary. Reusing results from one project to the other should be possible.

We must also mention the work of Krauss *et al* on the formalization in the proof assistant Isabelle/HOL [34] of a termination criterion using lexicographic comparisons [6] and of Lee, Jones and Ben-Amram's size-change principle [29]. The motivation of Krauss *et al* is slightly different than ours. They developed implementations of these termination criteria that directly produce Isabelle/HOL proofs, in order to automatically check the termination of Isabelle/HOL functions. But it should be possible for any termination prover using these criteria to produce some certificate from which a proof in Isabelle/HOL could be built.

7 Evaluation

The termination competition [1] is a forum for termination provers to compete on a set of problems, the so called termination problems database [2]. It allows to compare different tools and techniques and stimulates the research in the area of automated termination. In this competition every tool is run on every problem from the database and for every TRS, unless it gives up, it must decide whether it is (non-)terminating and support this claim with an informal textual description, which, in principle, should provide enough information for a human to reconstruct the complete termination proof.

The competition started back in 2003 but in 2007 for the first time the certified category was introduced, showing recognition for the importance of the work in this area. In this category every tool must support its claim with a full proof checkable in some well established proof assistant/checker. This ensures the highest reliability of the results one can get.

In the first edition of the competition featuring this new category there were three participants. Two termination provers, TPA [26] and $\mathsf{T_TT_2}$ [21], used the combination of CoLoR/Rainbow as the certification back-end; the third one, C*i*ME [8], was instead relying on the Coccinelle library [7] for that purpose.

[4] The files generated by the tools and the time taken by Coq for checking them can be consulted on [1].

As for the results (that are available on [1]), the winner was the combination TPA+CoLoR that was able to prove termination of 354 out of the total 975 TRSs used in the competition. C*i*ME+Coccinelle came in second with the score of 317 and $\mathsf{T_TT_2}$+CoLoR took third spot with 289 points. Both CoLoR and Coccinelle changed since the time of the competition, however a detailed comparison is difficult as the latest version of Coccinelle is not publicly available yet.

TPA sets certification as one of its prior goals. At the moment TTPA (Trusted-TPA), a completely new, open source successor of TPA is under development. It is going to focus on certification by means of a close collaboration with the CoLoR project. But our approach is by no means limited to TPA. Already in the competition $\mathsf{T_TT_2}$ was able to successfully use CoLoR to certify its proofs. Recently, after addition of arctic interpretations to CoLoR, Matchbox [37] was able to certify 394 proofs (run on the exact problem set used for the 2007 competition). At the moment some experiments with certification of AProVE [18] (winner of the 2007 competition in the standard rewriting category) proofs are in progress and the initial results are encouraging.

8 Conclusion and Future Work

In the area of term rewriting, termination of first-order TRSs is an important topic, attracting a lot of research interest. There is a number of tools automatically producing proofs of termination. The complexity of those proofs is continuously increasing, which naturally calls for some way of verification. This has been recognized in the community and in 2007 in the annual termination competition a new certified category has been introduced.

In this paper we presented an approach to certification of termination proofs. It consists of a Coq library of formalized, termination related results and techniques (CoLoR), a formal grammar for termination certificates (TCG) and a simple program transforming proofs in the TCG format to formal Coq proofs (Rainbow).

This approach turned out to be successful and in the last certified termination competition the combination of CoLoR and TPA took the first spot. Also by now, apart from TPA, some proofs produced by $\mathsf{T_TT_2}$, Matchbox and AProVE were certified using the tools and methodology described in this paper.

So far two workshops on certified termination have been organized (Nancy in 2007 and Leipzig in 2008, see `http://termination-portal.org`) bringing together developers of automated termination provers and developers of certified libraries on termination. Based on the resulting discussions, we expect to carry on with the CoLoR project in various directions:

– Formalize, or finish to formalize, other transformational methods and termination criteria in CoLoR like RPO [11], the over-approximation of the dependency pairs graph based on unification [3], or semantic labeling [38].
– Add to CoLoR other notions of rewriting like innermost and AC rewriting.
– Try to improve the efficiency of the Coq functions used for computing the arguments and checking the conditions of termination criteria. Although, in

the last competition, the average time for Coq to check a proof generated by Rainbow was about 2 seconds, few proofs required substantially more time to be verified.

Acknowledgments. We thank everybody who contributed to the CoLoR library: Solange Coupet-Grimal, William Delobel, Léo Ducas, Sébastien Hinderer, Stéphane Le Roux, Johannes Waldmann and Hans Zantema.

References

1. Termination competition. `www.lri.fr/~marche/termination-competition`.
2. Termination problems data base. `http://www.lri.fr/~marche/tpdb`.
3. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *TCS*, 236(1-2):133–178, 2000.
4. Y. Bertot and P. Castéran. *Coq'Art: The Calculus of Inductive Constructions*. EATCS Texts in Theoretical Computer Science. Springer, 2004.
5. S. Briais. *Theory and tool support for the formal verification of cryptographic protocols*. PhD thesis, EPFL, Switzerland, 2008.
6. L. Bulwahn, A. Krauss, and T. Nipkow. Finding lexicographic orders for termination proofs in Isabelle/HOL. In *Proc. 20th TPHOL*, volume 4732 of *LNCS*, pages 38–53, 2007.
7. É. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Certification of automated termination proofs. In *Proc. 6th FroCoS*, volume 4720 of *LNAI*, pages 148–162, 2007.
8. É. Contejean, C. Marché, B. Monate, and X. Urbain. The CiME rewrite tool. Available at `http://cime.lri.fr`.
9. S. Coupet-Grimal and W. Delobel. An effective proof of the well-foundedness of the multiset path ordering. *AAECC*, 17(6):453–469, 2006.
10. D. Delahaye. A tactic language for the system Coq. In *Proc. 7th LPAR*, volume 1955 of *LNCS*, pages 85–95, 2000.
11. N. Dershowitz. Orderings for term-rewriting systems. *TCS*, 17:279–301, 1982.
12. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science*, volume B, chapter 6. North-Holland, 1990.
13. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *JAR*, 40(2–3):195–220, 2008.
14. C. Fuhs, R. Navarro, C. Otto, J. Giesl, S. Lucas, and P. Schneider-Kamp. Search techniques for rational polynomial orders. In *Proc. 9th AISC*, 2008. To appear.
15. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.
16. J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated termination analysis for haskell: From term rewriting to programming languages. In *Proc. 17th RTA*, volume 4098 of *LNCS*, pages 297–312, 2006.
17. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. 11th LPAR*, volume 3452 of *LNCS*, 2004.
18. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In *Proc. 15th RTA*, volume 3091 of *LNCS*, pages 210–220, 2004.
19. J. S. Golan. *Semirings and their Applications*. Kluwer, 1999.

20. S. Hinderer. Certification des preuves de terminaison par interprétations polynomiales. Master's thesis, Université Henri Poincaré, Nancy, France, 2004.
21. N. Hirokawa and A. Middeldorp. Tyrolean termination tool. In *Proc. 16th RTA*, volume 3467 of *LNCS*, pages 175–184, 2005.
22. INRIA Rocquencourt, France. *The Coq Proof Assistant Reference Manual, Version 8.1*, 2006. `http://coq.inria.fr/`.
23. J.-P. Jouannaud and A. Rubio. The Higher-Order Recursive Path Ordering. In *Proc. of the 14th IEEE Symposium on Logic in Computer Science*, 1999.
24. A. Koprowski. Certified higher-order recursive path ordering. In *Proc. 17th RTA*, volume 4098 of *LNCS*, pages 227–241, 2006.
25. A. Koprowski. Coq formalization of the higher-order recursive path ordering. Technical Report CSR-06-21, Eindhoven University of Technology, 2006.
26. A. Koprowski. TPA: Termination proved automatically. In *Proc. 17th RTA*, volume 4098 of *LNCS*, pages 257–266, 2006.
27. A. Koprowski and H.Zantema. Certification of proving termination of term rewriting by matrix interpretations. In *Proc. 34th SOFSEM*, 2008. To appear.
28. A. Koprowski and J. Waldmann. Arctic termination ... below zero. In *Proc. 19th RTA*, volume 4533 of *LNCS*, 2008. To appear.
29. A. Krauss. Certified size-change termination. In *Proc. 23rd CADE*, volume 4603 of *LNCS*, pages 460–475, 2007.
30. K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In *Proc. 1st PPDP*, volume 1702 of *LNCS*, 1999.
31. D. S. Lankford. On proving term rewriting systems are noetherian. Technical Report Memo MTP-3, Louisiana Tech. Univ., 1979.
32. S. Lucas. Practical use of polynomials over the reals in proofs of termination. In *Proc. 9th PPDP*, pages 39–50, 2007.
33. M. T. Nguyen, J. Giesl, P. Schneider-Kamp, and D. De Schreye. Termination analysis of logic programs based on dependency graphs. In *Proc. 17th LOPSTR*, 2007.
34. T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. 2002.
35. C. Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In *Proc. 1st TLCA*, volume 664 of *LNCS*, pages 328–345, 1993.
36. S. Le Roux. Acyclicity and linear extendability: a formal and constructive equivalence. Technical Report RR-2007-14, LIP-ENS, France, 2007.
37. J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proc. 15th RTA*, volume 3091 of *LNCS*, pages 85–94, 2004. `http://dfa.imn.htwk-leipzig.de/matchbox`.
38. H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.

Appendix

In this appendix we illustrate the use of dependent types by presenting in more detail the formalization by the first author of the notions of cap and aliens of a term.

Example 6. Assume that Sig is the disjoint union of two sets C and D. Then, the cap of t is the biggest term (up to variable renaming) matched by t and whose symbols are all in C. The biggest subterms of t that are headed by a symbol of D are the alien subterms of t.

These notions are often used in the proofs of modularity results, *i.e.* when combining terms of different signatures, and are indeed used in the proof of the dependency pair theorem described later. This formalization provides a nice example of higher-order dependent function.

Let Cap be the set of dependent triples (k, f, v) such that:

- k : nat is the number of aliens,
- f : terms $k \rightarrow$ term is a function which, given a vector v of k terms, returns the cap of t with the i-th alien replaced by the i-th term of v,
- v : terms k is the vector of the k aliens.

Consider also the following auxiliary functions:

- sum : $\forall n$: nat, Caps $n \rightarrow$ nat be the function computing the total number of aliens of a vector cs of Cap's (k_i, f_i, v_i): sum $cs = k_1 + \ldots + k_n$;
- conc : $\forall n$: nat, $\forall cs$: Caps $n \rightarrow$ terms (sum cs) be the function concatenating all the alien vectors of a vector cs of Cap's (k_i, f_i, v_i): conc $cs = v_1 @ \ldots @ v_n$;
- Vmap_sum : $\forall n$: nat, $\forall cs$: Caps n, terms(sum cs) $\rightarrow$ terms n be the function that, given a vector of Cap's (k_i, f_i, v_i), breaks a vector of sum cs terms into n vectors w_i of size k_i, apply f_i to w_i and concatenate all the results: Vmap_sum $cs = f_1(w_1) @ \ldots @ f_n(w_n)$.

We can now define the function cap : term $\rightarrow$ Cap whose code is given in Figure 2. If t is a variable, then there is no alien and the cap of t the constant function equal to t. If t is headed by a symbol $f \in D$, then t is an alien and the cap of t is the first projection. And if t is headed by a symbol $f \in C$, then the aliens are obtained by concatening the aliens of each of the subterms of t, and the cap of t is obtained by using Vmap_sum.

Then, we can prove that, for all term t, if cap$(t) = (k, f, v)$, then $t = f(v)$. The usual definition of cap is obtained by applying f to fresh variables.

```
Fixpoint cap (t : term) : Cap :=
  match t with
    | Var x => mkCap (fun _ => t, Vnil)
    | Fun f ts =>
      let fix caps n (ts : terms n) {struct ts} : Caps n :=
        match ts in vector _ n return Caps n with
          | Vnil => Vnil
          | Vcons t n' ts' => Vcons (cap t) (caps n' ts') end
      in if condition f then
           mkCap (fun v => Vnth v (lt_O_Sn 0), Vcons t Vnil)
         else let cs := caps (arity f) ts in
           mkCap (fun v => Fun f (Vmap_sum cs v), conc cs) end.
```

Fig. 2. Definition of the cap and aliens of a term in CoLoR