INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Automated Verification of Termination Certificates

Frédéric Blanqui — Adam Koprowski

N° 6949

Juin 2009

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--6949--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
PARIS - ROCQUENCOURT

Automated Verification of Termination Certificates

Frédéric Blanqui* , Adam Koprowski**

Thème SYM — Systèmes symboliques
Équipe-Projet FORMES

Rapport de recherche n° 6949 — Juin 2009 — 21 pages

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 is done by formalizing the proofs of termination criteria used in 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.

Key-words: termination,certification,formalization,proof,Coq

* INRIA, FIT 3-604, Tsinghua University, Haidian District, Beijing 100084, China
** Radboud University Nijmegen, P.O. Box 9010, 6500 GL, Nijmegen, The Netherlands, and MLstate, 15 Rue Berlier, 75013 Paris, France

Centre de recherche INRIA Paris – Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex
Téléphone : +33 1 39 63 55 11 — Télécopie : +33 1 39 63 53 30

Vérification Automatique de Certificats de Terminaison

Résumé : 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 is done by formalizing the proofs of termination criteria used in 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.

Mots-clés : terminaison,certification,formalisation,preuve,Coq

1 Introduction

In this paper we are concerned with termination of first-order term rewriting systems (TRSs) [13]. It is an important and 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 opted for the latter approach.

Our first contribution is CoLoR: a rich library of termination related results (55000 lines), developed in the proof assistant/checker Coq [25]. 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, 29, 30], hence in this paper we focus on the ones that were not presented before. 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 the 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 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 sources of the Rainbow program and the CoLoR library, can be found on `http://color.inria.fr/`.

2 Preliminaries

We begin by briefly recalling a few basic notions and refer to [13] 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* (set $\mathcal{D}$) and *constructors* (set $\mathcal{C}$): 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, *i.e.*, $f(\ldots t_i \ldots) \to_{\mathcal{R}} f(\ldots t'_i \ldots)$ whenever $t_i \to_{\mathcal{R}} t'_i$, and substitution, *i.e.*, $t\sigma \to_{\mathcal{R}} t'\sigma$ whenever $t \to_{\mathcal{R}} t'$. We introduce two restrictions of $\to_{\mathcal{R}}$:

- $\xrightarrow{\epsilon}_{\mathcal{R}}$ where rewriting is allowed only at the root position (top steps) and
- $\xrightarrow{>\epsilon}_{\mathcal{R}}$ 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 $t_i \to t_{i+1}$ for all $i \in \mathbb{N}$. A TRS $\mathcal{R}$ is called terminating if $\mathsf{SN}(\to_{\mathcal{R}})$ holds.

A binary relation $\to_1$ is called *terminating relative to* a binary relation $\to_2$, written as $\mathsf{SN}(\to_1 \,/\, \to_2)$, if there is no infinite sequence $t_0, t_1, \ldots$ such that

- $t_i \to_1 t_{i+1}$ for infinitely many values of i, and
- $t_i \to_2 t_{i+1}$ for all other values of i.

We use the notation $\to_1 \,/\, \to_2$ to denote $\to_2^* \cdot \to_1$, the composition of $\to_2^*$ and $\to_1$. Then $\mathsf{SN}(\to_1 \,/\, \to_2)$ coincides with well-foundedness of $\to_2^* \cdot \to_1$.

3 General architecture

In this section we 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 [25, 4]. 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 program transforming termination certificates to formal termination proofs, verifiable by Coq (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

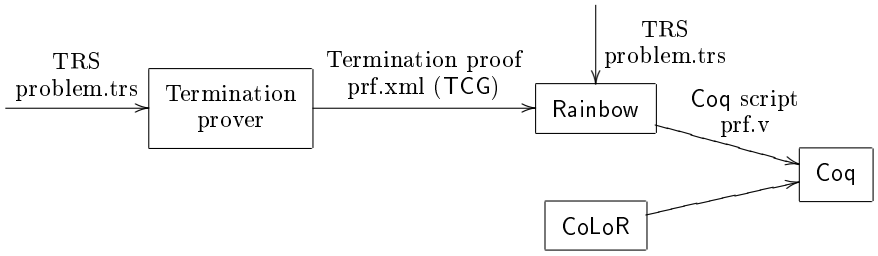

Fig. 1. Certifying termination with CoLoR and Rainbow

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 theorems and tactics of CoLoR. 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 [25, 4] is a proof assistant/checker based on the Calculus of Inductive Constructions (CIC) [37], a very rich typed λ-calculus following the proofs-as-objects principle including simple, inductive, dependent and polymorphic types. It allows to define functions using fixpoints and pattern matching, but recursive calls must be done on structurally smaller arguments to ensure termination. It also provides a high-level tactic language allowing to do non-linear pattern-matching on the current goals and hypotheses [11]. Proofs can then be built by using user-defined and built-in tactics ranging from basic tactics like applying a theorem, to complex tactics like a decision procedure for 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 embedding or a deep embedding. In a shallow embedding, no specific representation of the objects under consideration is developed in the proof assistant. For instance a polynomial is 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.

Altogether, the CoLoR library has nearly 55000 lines of code (including comments and blank lines) with approximately 1070 definitions, 130 tactics and 2800

theorems (many of which being simple but nonetheless necessary). As a comparison, the standard library of Coq 8.2 has approximately 127000 lines of code, 2660 definitions, 315 tactics and 7000 theorems.

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

Finally, it is important to note that all CoLoR theorems are proved constructively. In the literature on termination, 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/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 the functional type $A \to A \to \mathit{Prop}$, where *Prop* is the type of propositions. Strong normalization of an element x of type A is defined inductively as usual: SN R x if, for all y such that $R\ x\ y$, SN R y (this is equivalent to *Acc* (*transp* R) x in Coq). 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:

- As part of a more general work on Nash equilibrium, constructive proofs of various results on linear extensions [38].
- A function computing the strongly connected components of a finite relation (graph), using boolean adjacency matrices [14].
- 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* [21] 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 that is used for the inner workings

of the *ring* tactic. 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.

Vectors $vector : Type \to \mathbb{N} \to Type$ is the polymorphic and dependent inductive type whose constructors are $Vnil : \forall\ (A : Type), vector\ A\ 0$ and $Vcons : \forall\ (A : Type)\ (n : \mathbb{N}), A \to vector\ A\ n \to vector\ A\ (n+1)$. 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 they are quite powerful. For instance, a term of type $vector\ A\ n$ has n elements by construction. CoLoR provides a rich set of functions on vectors and theorems about them, that has been reused in a formalization of the spi-calculus [5].

Polynomials 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 [23]. 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 [34, 16].

Multisets Finite multisets have been formalized to prove the well-foundedness of the higher-order recursive path ordering (HORPO) [26]. The main property 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 are implemented as vectors of vectors and 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). Matrices are used for matrix and arctic interpretations [29, 30].

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 (SRS), term rewrite systems (TRS), 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 [35, 18]. 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: an SRS $\mathcal{R}$ terminates iff its inverse $\mathcal{R}^{-1} = \{(u, v) \mid (v, u) \in \mathcal{R}\}$ terminates.

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 $Var : \mathbb{N} \to term$ and $Fun : Sig \to list\ term \to term$. 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* and *Fun* defined below. This is the most 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. Below we will illustrate the basic notions of this term structure; another example can be found in Section A.

To define terms we will represent variables by natural numbers, so $\mathcal{V} = \mathbb{N}$ and the signature Σ is defined as:

Notation $variable := \mathbb{N}$ (*only parsing*).
Record $Signature : Type := mkSignature$ {
  $symbol :> Type$;
  $arity : symbol \to \mathbb{N}$;
  $eq\_symbol\_dec : \forall\ f\ g : symbol,\ \{f = g\} + \{\sim f = g\}$}.

So it is a set of symbols, *symbol*, an arity function, *arity*, and a procedure to decide equality of symbols. So now we can easily define terms, where a term is either a variable or a function symbol f from *Sig* applied to *arity* f terms.

Variable $Sig : Signature$.
Inductive $term : Type :=$

| *Var* : *variable* → *term*
| *Fun* : ∀ *f* : *Sig*, *vector term* (*arity f*) → *term*.

Now a rewrite rule is simply a pair of terms and a TRS is a list of rules. Note that for a rewrite rule $\ell \to r$ we do not enforce at this point the usual condition that ℓ is not a variable and the variables of r are a subset of variables of ℓ.

Record *rule* : *Type* := *mkRule* {*lhs* : *term*; *rhs* : *term*}.
Definition *trs* := *list rule*.

Simply-typed λ-terms Finally, CoLoR also provides a formalization of simply-typed λ-terms, using de Bruijn representation for bound variables. This formalization was used in the proof of well-foundedness of HORPO [26]. 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 subject reduction and termination for β-reduction. For more details we refer to [27].

4.4 Termination Results

The focus of CoLoR is on providing formalizations of actual methods for proving termination. At the moment we can handle 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 Currently CoLoR contains the following interpretation-based termination criteria: polynomial interpretations [33, 23], matrix interpretations [15, 29], and arctic interpretations [30].

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 of monotone algebras.

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 [19].

Now we describe the basic dependency pair transformation. For every defined symbol $f \in \mathcal{D}$ 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 $\mathcal{R}$, $g(t_1, \ldots, t_m)$ is a subterm of r and $g \in \mathcal{D}$, 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 first main result concerning the dependency pair transformation that has been formalized in CoLoR is:

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

For proving this theorem, we used the notion of (constructor) cap and (defined) aliens described in Appendix A. Indeed, a term is strongly normalizing if all its (defined) aliens are strongly normalizing.

Then, 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 [3]). *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 and 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 ([17]). *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 $i \in \{1, \ldots, n\}$.*

Certifying the use of such a technique raises two important difficulties.

First, 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 (this is the notion of cap described in Appendix A) and then replacing all variables of such a term with fresh ones (linearization).

To answer this first problem, we formalized in Coq some unification algorithm and proved its correctness, termination and completeness (the algorithm always terminates with a solution when two terms are unifiable). For proving its termination, we used the lexicographic and multiset orderings already formalized in CoLoR. Then, using the completeness of this unification algorithm, we could prove the correctness of the over-approximation just described.

The second problem is the computation of strongly connected components. In a first attempt, Léo Ducas formalized in CoLoR an algorithm for computing strongly connected components by using adjacency matrices [14]. It appeared that this was not very efficient in Coq. We then realized that we do not need to do such computations and formalized the following theorem in CoLoR:

Theorem 4. *Let $\mathcal{R}$ be a TRS, and $[C_1; \ldots; C_n]$ be a* valid decomposition, i.e.*:*

- $\bigcup_{i=1}^n C_i = \mathsf{DP}(\mathcal{R})$,
- *for all $i < j$, $x_i \in C_i$ and $x_j \in C_j$, there is no edge in $G(\mathcal{R})$ from x_j to x_i.*

Then, $\mathsf{SN}(\overset{\epsilon}{\to}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}})$ *iff* $\mathsf{SN}(\overset{\epsilon}{\to}_{C_i}/\to_{\mathcal{R}})$ *for every $i \in \{1, \ldots, n\}$.*

Hence, now, Rainbow can certify termination proofs based on this more general theorem. The first condition is reduced to testing the inclusion of the elements of a list into another one, which can be carried out efficiently in Coq by using the tactic "intuition". The second condition is reduced to boolean computations and the use of the unification algorithm, which can be carried out efficiently in Coq by using the tactic "vm_compute" [22].

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 5 (Argument filtering [32]). *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:*

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

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

Theorem 6 ([3]). *Let $\mathcal{E}$ and $\mathcal{R}$ be TRSs over a signature Σ and π be an argument filtering function on Σ. Then,* $\mathsf{SN}(\xrightarrow{\epsilon}_{\pi(\mathcal{R})} / \to_{\pi(\mathcal{E})})$ *implies* $\mathsf{SN}(\xrightarrow{\epsilon}_{\mathcal{R}} / \to_{\mathcal{E}})$.

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. It should not be too difficult to extend the formalization in order to get rid of that restriction.

Finally, it is interesting to note that the formalization of termination criteria represents only 19% of the size of CoLoR: 26% is about data structures, 14% is about mathematical structures and 41% is about term structures.

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. For instance, 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 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.

Below we present an example of a TRS, its termination proof expressed in the TCG format and the resulting Coq termination proof, as generated by Rainbow.

Example 7. Consider the rewrite system AG01/#3.1.trs from TPDB [1]:

$$\begin{array}{rr} \mathrm{minus}(x,0) \to x & \mathrm{minus}(\mathrm{s}(x),\mathrm{s}(y)) \to \mathrm{minus}(x,y) \\ \mathrm{quot}(0,\mathrm{s}(y)) \to 0 & \mathrm{quot}(\mathrm{s}(x),\mathrm{s}(y)) \to \mathrm{s}(\mathrm{quot}(\mathrm{minus}(x,y),\mathrm{s}(y))) \end{array}$$

Using the DP method introduced in Section 4.4 we obtain 3 dependency pairs:

(1) $\mathrm{minus}_\#(\mathrm{s}(x),\mathrm{s}(y)) \to \mathrm{minus}_\#(x,y)$ (2) $\mathrm{quot}_\#(\mathrm{s}(x),\mathrm{s}(y)) \to \mathrm{minus}_\#(x,y)$

(3) $\mathrm{quot}_\#(\mathrm{s}(x),\mathrm{s}(y)) \to \mathrm{quot}_\#(\mathrm{minus}(x,y),\mathrm{s}(y))$

and 2 SCCs in the approximated dependency graph: {(1)} and {(2),(3)}. Both SCC can be proven terminating using polynomial interpretations [33]; the latter one with the following interpretation:

$$\begin{aligned} \mathrm{s}(x) &= x+2 & \mathrm{minus}(x,y) &= x+1 \\ \mathrm{quot}_{\#}(x,y) &= xy+x+y & \mathrm{minus}_{\#}(x,y) &= x+1 \end{aligned}$$

To deal with this proof formally Rainbow first transforms the TPDB specification of the TRS in the TCG problem format (see Figure 3 in the appendix). Then the proof sketched above needs to be represented in the TCG proof format (see Figure 4 in the appendix for the snippet of this proof; note that the format is quite verbose but it is not intended to be directly read/written by humans).

Taking those two files as an input Rainbow automatically produces a Coq script containing formal termination proof. Such script corresponding to the proof sketched above is presented in Figure 2.

6 Related Work

In [7, 10], it is described another termination proof certification back-end based on a Coq library called Coccinelle developed for certifying the results of the automated termination prover C*i*ME [8]. The approach followed in this work is different from ours. C*i*ME uses shallow embeddings for representing rules, dependency pairs or polynomial interpretations, where as we use deep embeddings. They use no general theorem like the ones described in the previous sections, the conditions of which can be checked by doing simple computations and applying simple tactics. Instead, some adhoc proofs are generated each time. Hence, the size of Coccinelle is about half of the size of CoLoR, and the Coq scripts generated by C*i*ME are much longer than the ones generated by CoLoR.[1] Debugging a tool generating such long and complex Coq scripts must be difficult. In contrast, the scripts generated by Rainbow are short and clear: first, all proof ingredients (rules, interpretations, etc.) are defined; second, in the termination proof itself, each TCG proof node is translated by a tactic defined in CoLoR. Hence, even non Coq-experts can understand the generated files and, in case of failure, easily and precisely localize which theorem application did not succeed and why. See Appendix 7 for an example of generated proof.

We must also mention the work of Krauss *et al* on the formalization in the proof assistant Isabelle/HOL [36] of a termination criterion using lexicographic comparisons [6] and of Lee, Jones and Ben-Amram's size-change principle [31]. The motivation of Krauss *et al* is 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.

[1] Generated files and times taken by Coq for checking them can be consulted in [2].

```
Module M.(* the set of symbols in the signature *)
  Inductive symb : Type :=
  | minus : symb
  (* ... remaining symbols ... *).
End M.
Definition ar (s : M.symb) : ℕ := (* their arity *)
  match s with
  | M.minus ⇒ 2
  (* ... remaining symbols ... *)
  end.
Definition S := nil.(* 𝒮 is empty *)
Definition R := (* ℛ contains the set rewrite rules *)
  R0 (S0.minus (S0.succ (V0 0)) (S0.succ (V0 1)))
     (S0.minus (V0 0) (V0 1))
:: (* ... remaining rules ... *)
(* and we consider the rewrite relation →ℛ/→𝒮 *)
Definition rel := ATrs.red_mod S R.
(* polynomial interpretation used to solve SCC {(2),(3)} *)
Module PIS2 <: TPolyInt.
  Definition trsInt f :=
    match f as f return poly (@ASignature.arity s1 f) with
    | (hd_symb M.minus) ⇒ (* minus# mapped to x+1 *)
      ((1) % Z, (Vcons 1 Vnil))
      :: ((1) % Z, (Vcons 0 Vnil))
      :: nil
    (* ... remaining symbols ... *)
    end.
  (* Interpretation is weakly monotone *)
  Lemma trsInt_wm : ∀ f, pweak_monotone (trsInt f).
  Proof.pmonotone.Qed.
End PIS2.
Module PI2 := PolyInt PIS2.
(* termination proof of the system *)
Lemma termination : WF rel.
Proof.
dp_trans.                (* apply DP transformation *)
mark.                    (* mark DP symbols *)
graph_decomp(* ... *). (* graph decomposition *)
  dpg_unif_N_correct.  (* proof that decomposition is correct *)
  (* ... *)             (* proof for the first SCC *)
  PI2.prove_termination.(* proof for the second SCC with PI2 *)
  termination_trivial.   (* then termination trivial *)
Qed.
```

Fig. 2. Formal Coq script corresponding to the termination proof of Example 7

7 Evaluation

The termination competition [2] is a forum for termination provers to compete on a set of problems, the so called termination problems database (TPDB) [1]. 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 problem, unless it gives up, it must decide whether it is terminating or not 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 database contain string rewrite systems (SRS), term rewrite systems (TRS), logic programs and Haskell programs, and some of the termination problems are open.

The competition started back in 2003 and, in 2007, for the first time the certified TRS category was introduced, showing recognition for the importance of the work in this area. Indeed, every year, several tools are disqualified because some error is found in their answers. In the certified 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.

It is difficult to make a precise, fair comparison between CoLoR and A3PAT (*i.e.*, the C*i*ME certification tool based on Coccinelle), all the more so since they do not support the same termination techniques. For instance, Coccinelle supports lexicographic path ordering and graph decomposition based on unification (CoLoR supports this now too), while CoLoR supports matrix interpretation. We however summarized the results of the last two competitions hereafter. Note that AProVE-cert runs AProVE-CoLoR and AProVE-A3PAT in parallel.

2008 TRS competition		
Tool	Score/1391	%
AProVE [20]-cert	594	42.7
AProVE-CoLoR	580	41.7
AProVE-A3PAT	532	38.2
C*i*ME [8]-A3PAT	531	38.2
Matchbox [39]-CoLoR	458	32.9

2008 SRS competition		
Tool	Score/732	%
Matchbox [39]-CoLoR	466	63.7
AProVE [20]-cert	420	57.4
AProVE-CoLoR	415	56.7
AProVE-A3PAT	114	15.6

2007 TRS competition		
Tool	Score/975	%
TPA [28]-CoLoR	354	36.3
C*i*ME [8]-A3PAT	317	32.5
T_TT_2 [24]-CoLoR	289	29.6

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 program transforming proofs in the TCG format to formal Coq proofs (Rainbow). Rainbow is kept as simple as possible in order not to introduce errors.

This approach turned out to be successful since CoLoR-Rainbow (now used by four different provers: AProVE, Matchbox, TPA and T_TT_2) was the best certification back-end in the last two termination competitions [2].

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 like RPO [12] or semantic labeling [40].
- Add to CoLoR other notions of rewriting like innermost and AC rewriting.
- Improve the efficiency of the functions used for computing the arguments and checking the conditions of termination criteria. Although, in the 2007 competition, the average time for Coq to check a proof 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, Jörg Endrullis, Sébastien Hinderer, Stéphane Le Roux, Johannes Waldmann and Hans Zantema.

References

1. Termination problems data base. `http://colo5-c703.uibk.ac.at:8080/termcomp/home.seam`.
2. `http://termination-portal.org/wiki/Termination_Competition`.
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.
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. P. Courtieu, J. Forest, and X. Urbain. Certifying a termination criterion based on graphs, without graphs. In *Proc. of TPHOL'08*, LNCS 5170, 2008.
11. D. Delahaye. A tactic language for the system Coq. In *Proc. 7th LPAR*, volume 1955 of *LNCS*, pages 85–95, 2000.
12. N. Dershowitz. Orderings for term-rewriting systems. *TCS*, 17:279–301, 1982.
13. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science*, volume B, chapter 6. North-Holland, 1990.
14. Leo Ducas. Certication of termination proofs based on dependency graph decomposition, 2007. B.Sc. thesis.
15. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *JAR*, 40(2–3):195–220, 2008.
16. 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.
17. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.
18. 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.
19. 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.
20. 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.
21. J. S. Golan. *Semirings and their Applications*. Kluwer, 1999.
22. B. Grégoire and X. Leroy. A compiled implementation of strong reduction. In *Proceedings of the 7th ACM International Conference on Functional Programming*, SIGPLAN Notices 37(9), 2002.
23. S. Hinderer. Certification des preuves de terminaison par interprétations polynomiales. Master's thesis, Université Henri Poincaré, Nancy, France, 2004.
24. N. Hirokawa and A. Middeldorp. Tyrolean termination tool. In *Proc. 16th RTA*, volume 3467 of *LNCS*, pages 175–184, 2005.
25. INRIA Rocquencourt, France. *The Coq Proof Assistant Reference Manual, Version 8.1*, 2006.
26. A. Koprowski. Certified higher-order recursive path ordering. In *Proc. 17th RTA*, volume 4098 of *LNCS*, pages 227–241, 2006.
27. A. Koprowski. Coq formalization of the higher-order recursive path ordering. Technical Report CSR-06-21, Eindhoven University of Technology, 2006.
28. A. Koprowski. TPA: Termination proved automatically. In *Proc. 17th RTA*, volume 4098 of *LNCS*, pages 257–266, 2006.
29. A. Koprowski and H.Zantema. Certification of proving termination of term rewriting by matrix interpretations. In *Proc. 34th SOFSEM*, 2008. To appear.
30. A. Koprowski and J. Waldmann. Arctic termination ... below zero. In *Proc. 19th RTA*, volume 4533 of *LNCS*, 2008. To appear.
31. A. Krauss. Certified size-change termination. In *Proc. 23rd CADE*, volume 4603 of *LNCS*, pages 460–475, 2007.
32. K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In *Proc. 1st PPDP*, volume 1702 of *LNCS*, 1999.

33. D. S. Lankford. On proving term rewriting systems are noetherian. Technical Report Memo MTP-3, Louisiana Tech. Univ., 1979.
34. S. Lucas. Practical use of polynomials over the reals in proofs of termination. In *Proc. 9th PPDP*, pages 39–50, 2007.
35. 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.
36. T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. 2002.
37. C. Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In *Proc. 1st TLCA*, volume 664 of *LNCS*, pages 328–345, 1993.
38. S. Le Roux. Acyclicity and finite linear extendability: a formal and constructive equivalence. In *Proceedings of the 20th International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 4732, 2007.
39. J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proc. 15th RTA*, volume 3091 of *LNCS*, pages 85–94, 2004.
40. H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.

A Example of higher-order dependent program

We illustrate the use of higher-order dependent types by presenting the formalization of the notions of cap and aliens of a term used in CoLoR.

Assume that *Sig* is the disjoint union of two sets $\mathcal{C}$ and $\mathcal{D}$ (*e.g.* constructors and defined symbols). Then, the *cap* of t is the biggest term (up to variable renaming) matched by t and whose symbols are all in $\mathcal{C}$. The biggest subterms of t that are headed by a symbol of $\mathcal{D}$ 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 theorems described in Section 4. This formalization provides a nice example of higher-order dependent function.

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

- $k : \mathbb{N}$ is the number of aliens,
- $f : terms\ k \to term$ 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:

- $sum : \forall\ n, Caps\ n \to \mathbb{N}$ be the function computing the total number of aliens of a vector *cs* of *Cap*'s (k_i, f_i, v_i): $sum\ cs = k_1 + \ldots + k_n$;
- $conc : \forall\ (n : \mathbb{N})\ cs, Caps\ n \to terms\ (sum\ cs)$ be the function concatenating all the alien vectors of a vector *cs* of *Cap*'s (k_i, f_i, v_i): $conc\ cs = v_1 @ \ldots @ v_n$;
- $Vmap\_sum : \forall\ (n : \mathbb{N})\ (cs : Caps\ n), terms\ (sum\ cs) \to terms\ n$ 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: $Vmap\_sum\ cs = f_1(w_1) @ \ldots @ f_n(w_n)$.

Then the function *cap* defined below is such that, 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 (λ_ ⇒ 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 (λv ⇒ Vnth v (lt_O_Sn 0), Vcons t Vnil)
         else let cs := caps (arity f) ts in
           mkCap (λv ⇒ Fun f (Vmap_sum cs v), conc cs) end.
```

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 $f \in \mathcal{D}$, then t is an alien and the cap of t

is the first projection. And if t is headed by a symbol $f \in \mathcal{C}$, 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*.

B XML files in the TCG format

```
<problem>
 <trs>
  <algebra>
   <signature>
    <mapping>
     <fun>minus</fun>
     <arity>2</arity>
    </mapping>
    <!-- arity mapping for remaining symbols -->
   </signature>
  </algebra>
  <rules>
   <rule>
    <lhs>
     <app><fun>minus</fun>
      <arg><app><fun>s</fun>
            <arg><var>0</var></arg></app></arg>
      <arg><app><fun>s</fun>
            <arg><var>1</var></arg></app></arg>
     </app>
    </lhs>
    <rhs> <!-- righ-hand side of the rule --> </rhs>
   </rule>
   <!-- remaining rules -->
  </rules>
 </trs>
</problem>
```

Fig. 3. TCG description of the TRS introduced in Example 7

```
<proof>
 <dp>                          <!-- (1) apply DP transformation      -->
 <proof>                       <!--     proof after performing (1)  -->
  <mark_symbols>               <!-- (2) mark the DP symbols (#)      -->
   <proof>
    <decomp>                   <!-- (3) perform the graph            -->
     <graph>                   <!--     decomposition using graph   -->
      <unif/>                  <!--     approx. via unification     -->
     </graph>
     <component>               <!--(4a) proof for the first SCC      -->
       <...>                   <!--     omitted                     -->
     </component>
     <component>               <!--(4b) the second SCC               -->
      <rules>                  <!--     spec. of the component      -->
       <...>                   <!--     (list of its rules)         -->
      </rules>
      <proof>                  <!--     proof of its termination    -->
       <manna_ness>            <!--     we use Manna-Ness criterion -->
        <order>
         <poly_int>            <!--     with polynomial interpret.  -->
          <mapping>            <!--     we map:                     -->
           <fun>
            <hd_mark>          <!--     minus_#(x,y)                -->
             minus
            </hd_mark>
           </fun>
           <polynomial>        <!--     to the polynomial (1*x + 1) -->
            <monomial>         <!--     first monomial (1*x)        -->
             <coef>1</coef>    <!--       1*                        -->
             <var>1</var>      <!--       x^1*                      -->
             <var>0</var>      <!--       y^0                       -->
            </monomial>
            <monomial>         <!--     second monomial: (1)        -->
             <coef>1</coef>
             <var>0</var>
             <var>0</var>
            </monomial>
           </polynomial>
          </mapping>
          <...>                <!--     mapping of remaining symbols -->
         </poly_int>
        </order>
        <proof>                <!--     after PI there are no more  -->
         <trivial/>            <!--     rules so termination is     -->
        </proof>               <!--     trivial                     -->
       </manna_ness>
      </proof>
     </component>
    </decomp>
   </proof>
  </mark_symbols>
 </proof>
</dp>
</proof>
```

Fig. 4. Termination proof presented in Example 7 in the TCG format

Centre de recherche INRIA Paris – Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399