Automated Certified Proofs with CiME3*

É. Contejean[1], P. Courtieu[2], J. Forest[3], O. Pons[2], and X. Urbain[4]

1   CNRS, LRI, UMR 8623, Orsay, F-91405
     Univ Paris-Sud 11, Orsay, F-91405
     INRIA Saclay - Île-de-France, ProVal, Orsay, F-91893
     France
     `evelyne.contejean@lri.fr`
2   CNAM, Lab. Cédric, Paris, F-75141
     France
     `{pierre.courtieu,olivier.pons}@cnam.fr`
3   ENSIIE, Évry, F-91025
     Lab. Cédric, Paris, F-75141
     France
     `julien.forest@ensiie.fr`
4   ENSIIE, Évry, F-91025
     INRIA Saclay - Île-de-France, ProVal, Orsay, F-91893
     LRI, Univ Paris-Sud 11, CNRS, Orsay, F-91405
     France
     `xavier.urbain@ensiie.fr`

─── **Abstract** ───

We present the rewriting toolkit CiME3. Amongst other original features, this version enjoys two kinds of engines: to handle and discover proofs of various properties of rewriting systems, and to generate Coq scripts from proof traces given in *certification problem format* in order to certify them with a sceptical proof assistant like Coq. Thus, these features open the way for using CiME3 to add automation to proofs of termination or confluence in a formal development in the Coq proof assistant.

1 Introduction

Automated tools based on complex mathematical arguments have become widely used in various domains of computer science. Cryptographic systems, proof assistants or systems analysing programs involve highly intricate deduction procedures that are indeed beyond human capabilities. However, this situation raises a real issue as the downside of such deductive power is the difficulty to trust a result that no human can check.

Regarding rewriting, and in particular automated termination proof, there have been many new tools since the introduction of the dependency pair approach [1] at the end of the 90's: CiME 2, AProVE [13], TTT₂ [17], Jambox,[1] etc., to cite a few of them.


However, all these tools exhibited incorrect behaviour at some point, in particular during the Termination Competition [20] or its preparatory rounds. Several approaches to certify the results of automated provers with skeptical proof assistants have been developed: A3PAT [5, 4] and CoLoR/Rainbow [2] for Coq, CeTA [22] for Isabelle/HOL.

As a new member of the C*i*ME family (`http://cime.lri.fr`), C*i*ME3 is a toolbox dedicated to the handling and analysis of rewriting programs. It allows one to define term algebras, rewriting systems, etc., and to perform a range of treatments over them: computation, normalisation, matching and unification (modulo equational theories), completion (Knuth-Bendix) and proofs of equality (both modulo equational theories), etc. An important part of C*i*ME3 is dedicated to proofs of termination (including solving of ordering constraints), local confluence and convergence. C*i*ME3 enjoys a top level mode for interactive development of rewrite programs; it can also be used in batch mode with full automation.

C*i*ME3 has been developed within the A3PAT project for certification of automated proofs, and automation for proof assistants (`http://a3pat.ensiie.fr`). In comparison to its ancestors, C*i*ME3 features, in addition to its various proof engines, a certification mechanism which issues proof traces and *certificates*, and which allows one to check that the results are correct. To be certified, a proof trace can be translated into a script checked by a trusted tool: a skeptical proof assistant like Coq, or it can be given directly to a certified dedicated tool like CeTA. A unique feature of C*i*ME3 is in particular the (discovery and) certification of proofs of convergence.

The present article is organised as follows. In Section 2 we present some of the proof engines at work in C*i*ME3, for termination in Sections 2.1.2 and 2.1.3, and local confluence, Knuth-Bendix completion and convergence in Section 2.2. Some criteria allow for parallel solving, this is presented in Section 2.1.4. The certification engine is then described in Section 3, along with its input, output, and the general structure of emitted Coq scripts. Section 3.3 sketches the use of C*i*ME3 in batch mode. We conclude with a list of resources and companion tools, and with future work regarding the current implementation.
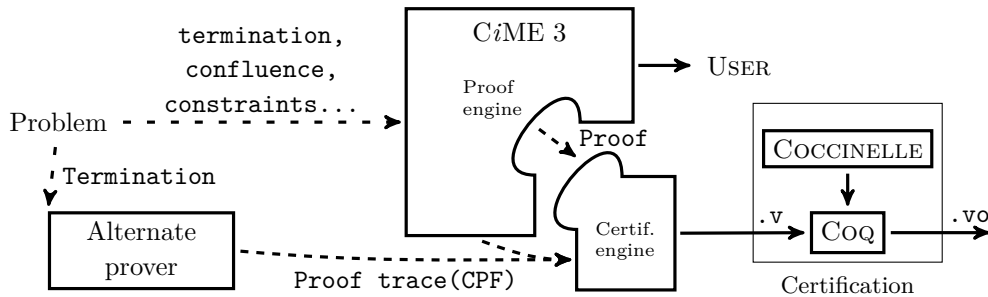
■ **Figure 1** Global C*i*ME 3 architecture.

2   Proof Engines

2.1   Core Language, Termination

C*i*ME3 can be used in interactive or batch mode. We hereafter describe some of the basic commands for the interactive mode. Note that the batch mode can accept additional files

in various formats (like `TRS` or `SRS` formats from the Termination Competition, or in CPF[2] (an XML format widely accepted by the certified rewriting community), developed by the CETA group and our team), however the commands described here can be used in both modes. See Section 3.3 for batch specific command line options.

The core top level language of CiME3 is a simple language which allows the user to type in expressions terminated by a semicolon. Simple expressions consist of booleans, integers, strings, and basic operations on them. As in previous versions of CiME the nominal language is powerful enough to define functions (polymorphic higher-order, allowing partial application, functions as arguments) with a `let fun` construct. Application is denoted by juxtaposition of the function and its arguments, as in LISP or other functional languages based on the λ-calculus. A short manual is available online on the A3PAT web page. We focus here on the new syntax and features of CiME3 for rewrite systems and constraints.

2.1.1 Declarations

The user can define and name objects, algebras, and systems in the CiME input language akin to the definitions found in the literature: sets of variables, signatures (symbols with arities), term algebras, rewrite systems, term ordering constraints...

A `let` construct introduces *global* declarations. Listing 1 declares successively the global set `X` of variables, a signature `F` for Peano numbers and addition, and the corresponding term algebra `T`. One can then easily declare terms, rewrite systems, and even ordering constraints (built from conjunctions `/\` and disjunctions `\/` of atomic constraints).

■ **Listing 1** A few simple declarations.

```
let X = variables "x,y";
let F = signature "plus : binary; O:constant; S:unary;";
let T = algebra F;
let t1 = term T "S(O)";
let R = trs T " plus(O,x) -> x; plus(S x, y) -> S(plus(x,y)); ";
let c = order_constraints T "O < S(O) /\ S(plus(x,y)) < plus(S(x),y)";
```

2.1.2 Ordering Parameters

CiME3 features a proof engine dedicated to the discovery of (well-founded) term orderings that fulfil a set of (ordering) constraints. These constraints may come from a direct declaration by the user (see Listing 1), or may appear as the result of operations like the application of termination criteria on a termination problem, i.e. from termination constraints. As soon as the ordering constraints are provided, one can try to discover a relevant well-suited ordering pair $(\preceq, <)$ using the command `ordering_solve` or `ordering_solve_strict` depending on the expected monotonicity of the ordering, that is depending on whether only $\preceq$, or both $\preceq$ and $<$, respectively, are supposed to be monotonic.

■ **Listing 2** Ordering constraints solving commands.

```
ordering_solve c;    (* Search for weakly monotonic well-founded ordering fulfilling c *)
ordering_solve_strict c;      (* Search for strictly monotonic well-founded ordering *)
```

---

[2] The Certification Problem Format, `http://cl-informatik.uibk.ac.at/software/cpf/`

C*i*ME3 can search for different kinds of orderings, all of which being parameterised by the user. The presently implemented orderings range from full RPO [10], with or without argument filterings (AFS) [1], to various polynomial [19, 18, 7] and matrix [11, 9] interpretations. Note that the *certification* engine can accept traces from other provers, and, thus, can handle more orderings than those that are currently searched for by the termination proof engine.

The parameters for the search are provided using the grammar in Figure 2. Ordering specifications (*Ordering_param*) are of the generic form n_1 *ord_kind* n_2 n_3 n_4... where *ord_kind* specifies the kind of ordering for which to search, n_1 is the (optional) timeout of the solver in seconds, n_2 (only for polynomial and matrix interpretations) specifies the upper bound of the (non-negative integral) coefficients in polynomials or matrices, n_3 (matrices only) specifies the size of the matrix coefficients, and n_4 and following parameters are the list of allowed sizes of strict sub-matrices as defined in [9]. For example, `matrix 1 3 1 2` specifies a search for matrix interpretations, with matrix coefficients less or equal to 1, 3×3 matrices, and strict sub-matrices of size either 1 *or* 2. As the optional timeout is not provided, the default will be used (no timeout). Similarly, `30 linear 3` specifies a search for linear polynomials with coefficients less or equal to 3, and a search timeout after 30s.

The user can declare several ordering parameters for different situations. This is illustrated in Listing 3 where two different ordering parameter sequences `op1` and `op2` are declared, in order to be used later in heuristics (see Listing 4).

■ **Listing 3** Ordering parameters.

```
let op1 = params "30 linear 3; 30 simple 2; 30 rpo; 30 matrix 2 2 1; 30 matrix 1 3 1 2";
let op2 = params "linear 3; simple 2; rpo; matrix 2 2 1; matrix 1 3 1";
```

Ordering parameters are interpreted sequentially, unless a parallel search is enabled. In the latter case, several orderings are searched for at the same time, depending on the computer architecture and the specific parameters given by the user, see Section 2.1.4.

2.1.3 Criteria and Heuristics

Termination criteria are commonly seen as transformations of termination problems into sets of other termination problems (possibly empty); they can be represented as inference rules [5, 8] or processors [14]. The strategy of application of termination criteria is specified in C*i*ME3 with the help of *heuristics*. Intuitively, a successful heuristic must describe a cover of a closed termination proof tree.

C*i*ME comes with a simple heuristic description language which contains built-in criteria and criteria combinators. Heuristics are specified following the grammar described in Figure 2 where: *id* is a previously declared heuristic, `manna_ness` stands for the well-known Manna and Ness criterion [19], and `lex_manna_ness` for rule removal using lexicographic combination of orderings; `DP` and `DPM` stand respectively for unmarked and marked dependency pair criteria [1], `DPG` for the graph refinement (estimation EDG) of dependency pairs [1] (`DPG` returns the set of strongly connected components); `ST` denotes the extension [4] of the original subterm criterion [16]; `RMVx` denotes vertex removal in dependency graphs components [14, 15], and `RMC` stands for its variant removing all vertices of the component.

A heuristic x applies to a problem and returns the set of generated sub-problems. The following constructs allow one to combine heuristics. `Then[` $x;y...$ `]` applies heuristic x on a problem and then, if it succeeds, heuristic y on each generated sub-problem. It fails if either x or y fails. `Repeat` x calls x on a problem and, if x succeeds, applies `Repeat` x again

$$
\begin{array}{lll}
\mathit{Ordering\_param} & ::= & \texttt{linear}\ n \mid \texttt{simple}\ n \mid \texttt{quadratic}\ n \mid \texttt{rpo} \mid \texttt{matrix}\ n\ n\ n+ \\
\mathit{Solver\_params} & ::= & n\ \mathit{Ordering\_param}\ (\texttt{;}\ \mathit{Solver\_params})? \\
\mathit{Criterion} & ::= & \texttt{manna\_ness} \mid \texttt{lex\_manna\_ness} \mid \texttt{DP} \mid \texttt{DPM} \mid \texttt{DPG} \mid \texttt{ST} \mid \texttt{RMVx} \mid \texttt{RMC} \\
\mathit{Heuristic} & ::= & \mathit{Criterion}\ \{\mathit{Solver\_params}\}? \mid \mathit{id}\ \{\mathit{Solver\_params}\}? \\
& & \mid\ \texttt{Then}\ [\mathit{Heuristic\_l}] \mid \texttt{Do}\ n\ \mathit{Heuristic} \mid \texttt{Repeat}\ \mathit{Heuristic} \mid \texttt{Solve}\ [\mathit{Heuristic\_l}] \\
\mathit{Heuristic\_l} & ::= & \mathit{Heuristic}\ (\texttt{;}\ \mathit{Heuristic\_l})?
\end{array}
$$

■ **Figure 2** Grammar for heuristics and ordering parameters.

recursively on sub-problems. It never fails. Do n x is similar to `Repeat` but uses a limit n for recursion depth. Finally `Solve [ ` x`;`$y\ldots$ ` ]` tries to apply x, and in the case x does not succeed tries to apply y, etc., it fails if all heuristics failed. When all recursive calls have ended, the proof is successful if all sub-problems are empty.

Listing 4 provides two examples. Heuristic `h2` first tries to apply repeatedly rule removal by lexicographic combination (using strictly monotonic orderings) until it fails to do so. It transforms then the remaining TRS termination problem into a marked dependency pairs termination problem and repeats recursively the previously declared `h1`. Heuristic `h1` first splits the given (DP) problem into sub-problems corresponding to the strongly connected components of the dependency graph, and then tries on each component: firstly the extended subterm criterion, and then, if it failed, vertex removal with a weakly monotonic ordering.

Some criteria may be parameterised specifically, for instance the number of rewrite steps allowed in the extended subterm criterion may be set to n using command `subtermparams` n.

■ **Listing 4** Heuristics declarations and commands.

```
let h1 = heuristic " Repeat Then [ DPG ; Solve [ ST ; RMVx {op2} ] ] ";
let h2 = heuristic " Then [ Repeat lex_manna_ness {op1} ; DPM ; h1] ";
set_heuristic "h2";                         (* Sets the termination heuristic. *)
```

When relevant parameters are set, proof search may be launched using commands `termination`, `confluence`..., which print answers and possibly store traces.

■ **Listing 5** Proof search.

```
termination R;
local_confluence R;
convergence R;
complete o R;                           (* Completion of R using term ordering o *)
prove_goal o R t1 t2; (* Ordered completion of R, stops when t1 and t2 are found equal*)
unify (term T "plus(O,x)") (term T "plus(y,O)");
```

2.1.4 External Solvers, Parallel Search for Orderings

External Solvers   Ordering constraints are nowadays usually translated into SAT problems. Thus, the main solving parts are delegated to a SAT solver [12, 21] or an SMT solver [17]. This scheme is implemented in C*i*ME3, and the discovery of interpretations, RPO and AFS, and application of the extended subterm criterion amount to solving SAT problems.

The directive **#Set_sat_solver** *invoc_name* allows one to specify the invocation name of the external SAT solver, which must fulfil the requirements of the SAT format for input and output, and the invocation of which must be of the form: *invoc_name in_file out_file*.

Note that polynomial interpretations and LPO+AFS may still be found without the help of any external solver, using the internal Diophantine constraint solver of C*i*ME [7].

Parallel Search   Ordering solvers may be used in parallel. When ordering constraints are generated during termination analysis, C*i*ME3 forks one solver process by ordering constraints set and ordering parameter (Listing 3), thus searching for different orderings in parallel. Note that solving of different ordering constraints is thus also parallelised. The maximum number of processes that can be launched in parallel may be set using the `#Set_nb_proc` directive. If this limit is reached, new forks are put in a queue and wait for previous ones to stop before they can be activated. Default behaviour is sequential computation.

2.2   Local Confluence, Completion and Convergence

A noticeable feature of C*i*ME3 is its ability to check and moreover certify local confluence. Combining this with its possibility to prove and certify termination, one can easily obtain *proof and certification* of convergence. To date and to our knowledge C*i*ME3 is the only tool that can prove and certify convergence of rewrite systems. The proof search of local confluence (and hence of convergence) is obtained by checking joinability of critical pairs. When proving local confluence, C*i*ME3 stores a trace in order to be able to certify it later.

Since critical pairs computation is the core of the standard Knuth-Bendix completion, our implementation of Knuth-Bendix completion (commands `complete` or `prove_goal`) also benefits from the trace production: given two terms which are found equal thanks to ordered completion, C*i*ME3 yields a trace of equality between these terms, and can also produce a Coq certificate [3]. Completion modulo AC, also a part of C*i*ME3, is not instrumented yet.

Certification of convergence proofs is obtained by an application of Newman's lemma, either using a known proof of termination, or assuming termination of the considered system. Convergence is then proved by showing that in particular each critical pair is joinable: we try to normalise each member of the pair and to show that both reduce to the same term.

3   Certification Engine

As proof engines perform proof search, various verbosity levels allow one to control the process. Once a proof is discovered, its description is printed on the standard output. Generation of a CPF trace,[3] or directly of the corresponding Coq script (for termination, local confluence and convergence), is enabled by command or through batch mode (Section 3.3).

The purpose of the certification engine is to take a proof trace as an input, and to output a Coq script for this trace's certification. Traces may come from proofs discovered with C*i*ME3, or with other provers (AProVE, T$_T$T$_2$. . .) provided they come as CPF files[3].

3.1   Input, Output

The normal input *trace* format for termination proofs is CPF. C*i*ME3 is not yet able to certify all criteria supported by the CPF format, but may generate scripts for all criteria it uses in proof discovery. For properties other than termination (including termination for convenience), one can take as an input a *problem* instead of a proof trace. In this case all the problem formats described in Section 2.1 are supported, C*i*ME will search the proof by itself and will generate the script, without any intermediate trace. See Listing 7.

The techniques for the generation of Coq scripts for certification have been described in previous works [4, 6, 8]. The compilation of those scripts relies on the Coccinelle library,

---

[3] As CPF is not yet extended to handle confluence proofs, hence confluence and convergence proofs output Coq scripts instead of CPF.

◼ **Listing 6** Coq script structure (notations are simplified).

```
1  Require Import equational_theory...              (* Preamble: require coccinelle files. *)
2  Module algebra.
3    Module F <:term_spec.Signature.                          (* Signature definition *)
4      Inductive symb:Set := plus: symb | S: symb | O: symb.            (* Symbols *)
5      ...
6    End F.
7    Module Alg := term.Make(F)(IntVars)          (* Algebra by functor instantiation *)
8  End Algebra.
9
10 Inductive R_rules: term → term → Prop :=                   (* Definition of the TRS *)
11 |R_rule₀: R_rules (Var 1) (Term plus [Term O [];Var 1])             (* plus(O,x)→x *)
12 |R_rule₁: R_rules (Term S [Term plus [Var 1;Var 2]])    (* plus(S(x),y)→S(plus(x,y)) *)
13                   (Term plus [Term S [Var 1];Var 2]).
14
15 Module WF_R.
16   ...                                             (* Criteria application proofs *)
17   Lemma wf:  well_founded (one_step R_rules).          (* Main termination lemma *)
18     ...
19   Qed.
20 End WF_R.
21 Module Confluence := Newman.Confluence(...).
22 ...
23 Lemma confluence: ∀ x, Newman.confluence _ (one_step R_rules) x.
24   ...                                        (* Joinability of critical pairs.  *)
25 Qed.
```

which allows for deep and shallow embeddings of the theory of rewriting. Relying on a deep embedding to reuse generic theorems instead of reproving them for each instances is usually faster. However it is interesting to notice that C*i*ME does use both embeddings depending on the proofs performed. We claim in particular that avoiding an actual deep representation of the dependency graph is very efficient from a computational point of view, as presented in [8]. Listing 6 illustrates the fact that the formalisation is partly shallow: on the one hand, symbols and rewriting rules are defined by new inductive types (`symb` and `R_rules`), on the other hand, term structure and rewriting relations rely on generic deep definitions (`Term`, `Var` and `one_step`). However, depending on the content of the proof, a deep version of `R_rules`, automatically proved equivalent, is defined when needed.

3.2 Structure of a Proof

Listing 6 presents an excerpt of the proof of convergence for the example in Listing 1. It displays in particular the general structure of a proof script for certification using the A3PAT approach. Symbols (line 4) and algebra (line 7) are defined at the beginning of the file, followed by the definition of the system itself (line 10).[4] The main lemma for termination is at the end of the last termination related module: `WF_R`. The proofs related to the instances of the different criteria mentioned in the input trace are formalised and proved in a (nested) sequence of modules (from line 16), thus mirroring roughly the structure of proof tree described in [4].

---

[4] Notice that `R_rules t u` means that `u` rewrites to `t`.

Eventually, the confluence proof may be generated with or without an actual proof of termination (in the latter case, the confluence is proved *on the assumption* that the system terminates). As termination is shown in our example, there is no assumption in Listing 6.

To compile the script, and thus to ensure its validity, one must have the COCCINELLE library installed, and a shell variable COCCINELLE set to its root directory. The last lines of the generated COQ file contains the coqc command line to run.

3.3   Batch Mode Command Line Options

In batch mode C*i*ME takes an input file in a specified format, and returns a result file in a specified output format. The whole process may include proof searches and/or generation of COQ scripts. Options -term/-noterm and -confl/-noconfl specify which properties must be considered or not. Input options include: -itrs (-isrs) for TRS (SRS) (non XML) formats, -itptp for TPTP format,[5] -icime for C*i*ME language, and -icpf for CPF format. Similarly, output options include: -ocpf, -ocoq, and -ocime (outputs the global environment in C*i*ME format).

🟨 **Listing 7** Sample command lines.

```
1  cime -itrs foo.trs -term -ocpf foo.cpf
2  cime -icpf foo.cpf -ocoq foo.v
3  cime -itrs foo.trs -term -ocoq foo.v
4  cime -itrs foo.trs -term -confl -ocoq foo.v
5  cime -icime preamble.cim3 -itrs foo.trs -term -confl -ocoq foo.v
6  cime -icime preamble.cim3 -noterm -confl  -itrs foo.trs -ocoq foo.v
```

Line 1 asks for a termination proof for the system in file foo.trs and then generates its CPF trace into foo.cpf. Line 2 generates the COQ script foo.v to certify CPF trace foo.cpf. Line 3 is equivalent to line 1 followed by line 2 (no CPF file is generated). Line 4 additionally generates the COQ proof of confluence. Line 5 is similar to 4 but loads a preamble in C*i*ME format before proof search, this is a usual way to set parameters in batch mode. Line 6 produces a COQ confluence proof parameterised by a proof of termination (not discovered). All these options may finally be chained in one command line:

```
cime  -noconfl -itrs foo1.trs -ocpf foo1Term.cpf -confl -ocoq foo1TermConf.v\
      -icpf foo2.cpf -ocoq foo2Term.v -noterm -ocoq foo2Conf.v
```

This asks for a termination proof for the system in foo1.trs, generates the corresponding CPF file foo1Term.cpf, then produces (without any new search) a termination and confluence COQ proof foo1TermConf.v. It then generates the COQ script foo2Term.v to certify the CPF trace foo2.cpf, and build script foo2Conf.v to certify the confluence of the TRS in foo2.cpf. The last proof is parameterised by a proof of termination (not computed).

3.4   Experiments

The following tables present some experiments run on a 24GB machine equipped with 12 cores and running Linux. The base for termination is the category TRS (taken *raw*) of the TPDB 5.0 consisting of 2506 problems;[6] 11 simultaneous processes are allowed. Local confluence and convergence tests are run on the 1155 problems that are proved terminating within the global time limit, set to 180s. The version of COQ is the 8.3 release.

---

[5] www.tptp.org up to version 4.
[6] In particular strategies are *not* taken into account.

The vast majority of termination proofs are discovered within 10s, and 30% within $\frac{1}{10}$s. Regarding certification, the compilation of Coq script is slow compared to certification by extracted tools like CeTA. However, 730 termination proofs are certified within 180s, of which 78% within a minute. The only failures observed (out of this run) are due to time or memory limitations. Regarding local confluence and convergence, 310 proofs of convergence are discovered, 306 of which are certified within the time limit. The average compile time for convergence is less than 17s. Over 53% of the convergence proofs are certified within 10s.

Discovery	< 0.1s	< 1s	< 10s
Term. 1155	354	730	1012
Confl. 310	287	310	—

Certification	< 5s	< 10s	< 30s	< 60s
Term.	730	200	293	481	573
Confl.	306	157	234	287	299
Conv.	306	53	164	271	293

It is difficult to compare certifiers from the result of the certified termination competition. Firstly, CiME was the only certifier targeted to Coq participating in 2010. Secondly, the number of proofs that certifiers have validated essentially reflects the certifiers accordance to the provers' proving strategy, as they formalise criteria/orderings that are distinct.

4   Resources and Perspectives

CiME3 is an open source piece of software; it is available under the CeCiLL-C licence on the resources page of the A3PAT project: `http://a3pat.ensiie.fr/pub`. The web site provides ELF executables for 64 bit architectures, a tarball of the sources and installation instructions, as well as a short user-manual (`http://a3pat.ensiie.fr/pub/manual-cime3.html`), and a script tool to ease benches and tests over databases of problems. Note that it is also possible to give CiME3 a try online through a dedicated web interface with a limited choice of options at `http://a3pat.ensiie.fr/online`. The companion library COCCINELLE is also available from this page, and requires version `8.3` of the Coq proof assistant.

Perspectives regards proof and certification engines. Some proof discovery techniques implemented in CiME 2 have not been transferred yet, notably termination modulo equational theories and modular techniques, including usable rules refinements.

Regarding certification techniques for termination, a short term perspective is the handling of arctic matrices, min/max polynomials, usable rules, and proofs under strategies, as all the formal material is ready in COCCINELLE.

─── **References** ───────────────────────────────────

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

2   F. Blanqui, S. Coupet-Grimal, W. Delobel, S. Hinderer, and A. Koprowski. Color, a coq library on rewriting and termination. In A. Geser and H. Sondergaard, ed., *Extended Abstracts of the 8th Int. Workshop on Termination, WST'06*, Aug. 2006.

3   É. Contejean and P. Corbineau. Reflecting proofs in first-order logic with equality. In *20th Int. Conf. on Automated Deduction*, vol. 3632 of *LNAI*, pp. 7–22, Tallinn, Estonia, July 2005. Springer.

4   É. Contejean, P. Courtieu, J. Forest, A. Paskevich, O. Pons, and X. Urbain. A3PAT, an Approach for Certified Automated Termination Proofs. In *ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, pp. 63–72. ACM, 2010.

5 É. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Certification of Automated Termination Proofs. In B. Konev and F. Wolter, ed., *6th Int. Symp. on Frontiers of Combining Systems*, vol. 4720 of *LNAI*, pp. 148–162, Liverpool, UK, Sept. 2007. Springer.

6 E. Contejean, J. Forest, and X. Urbain. Deep-Embedded Unification. Technical Report 1547, Cédric lab., CNAM Paris, France, 2008.

7 É. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325–363, 2005.

8 P. Courtieu, J. Forest, and X. Urbain. Certifying a Termination Criterion Based on Graphs, Without Graphs. In C. Muñoz and O. Ait Mohamed, ed., *21st Int. Conf. on Theorem Proving in Higher Order Logics*, vol. 5170 of *LNCS*, pp. 183–198, Montréal, Canada, Aug. 2008. Springer.

9 P. Courtieu, G. Gbedo, and O. Pons. Improved Matrix Interpretation. In J. van Leeuwen, A. Muscholl, D. Peleg, J. Pokorný, and B. Rumpe, ed., *36th Conf. on Current Trends in Theory and Practice of Computer Science*, vol. 5901 of *LNCS*, pp. 283–295, Špindlerův Mlýn, Czech Republic, Jan. 2010. Springer.

10 N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, Mar. 1982.

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

12 C. Fuhs, A. Middeldorp, P. Schneider-Kamp, and H. Zankl. SAT Solving for Termination Analysis with Polynomial Interpretations. In *SAT 07*, vol. 4501 of *LNCS*, pp. 340–354, May 2007. Springer.

13 J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic Termination Proofs in the Dependency Pair Framework. In U. Furbach and N. Shankar, ed., *3rd Int. Joint Conf. on Automated Reasoning*, vol. 4130 of *LNCS*, pp. 281–286, Seattle, USA, Aug. 2006. Springer.

14 J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and Improving Dependency Pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.

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

16 N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: Techniques and Features. *Information and Computation*, 205(4):474–511, 2007.

17 M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. Tyrolean Termination Tool 2. In R. Treinen, ed., *20th Int. Conf. on Rewriting Techniques and Applications*, vol. 5595 of *LNCS*, pp. 295–304, Brasília, Brazil, July 2009. Springer.

18 D. S. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Mathematics Department, Louisiana Tech. Univ., 1979.

19 Z. Manna and S. Ness. On the termination of Markov algorithms. In *3rd Hawaii Int. Conf. on Systems Sciences*, pp. 789–792, Honolulu, USA, 1970.

20 C. Marché and H. Zantema. The Termination Competition. In F. Baader, ed., *18th Int. Conf. on Rewriting Techniques and Applications*, vol. 4533 of *LNCS*, pp. 303–313, Paris, France, June 2007. Springer.

21 P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving Termination Using Recursive Path Orders and SAT Solving. In B. Konev and F. Wolter, ed., *6th Int. Symp. on Frontiers of Combining Systems*, vol. 4720 of *LNAI*, pp. 267–282, Liverpool, UK, Sept. 2007. Springer.

22 R. Thiemann and C. Sternagel. Certification of Termination Proofs using CeTA. In T. Nipkow and C. Urban, ed., *22nd Int. Conf. on Theorem Proving in Higher Order Logics*, vol. 5674 of *LNCS*, pp. 452–468, Munich, Germany, Aug. 2009. Springer.