Tree Automata with Equality Constraints Modulo Equational Theories

Florent Jacquemard, Michaël Rusinowitch, Laurent Vigneron

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Tree Automata with Equality Constraints Modulo Equational Theories

Florent Jacquemard — Michael Rusinowitch — Laurent Vigneron

N° 5754

Novembre 2005

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--5754--FR+ENG

Tree Automata with Equality Constraints Modulo Equational Theories

Florent Jacquemard*, Michael Rusinowitch †, Laurent Vigneron ‡

Thème SYM — Systèmes symboliques
Projets CASSIS (UR Lorraine) et SECSI (UR Futurs)

Rapport de recherche n° 5754 — Novembre 2005 — 27 pages

Abstract: This paper presents new classes of tree automata combining automata with equality test and automata modulo equational theories. These tree automata are obtained by extending the standard Horn clause representations with equational conditions and rewrite systems. We show in particular that a generalized membership problem (embedding the emptiness problem) is decidable by proving that the saturation of tree automata presentations with suitable paramodulation strategies terminates. We sketch an application of these tree automata classes to the reachability problem for a fragment of the applied pi-calculus.

Key-words: First Order Theorem Proving, Tree Automata, Basic Paramodulation, Splitting, Verification of Infinite State Systems.

* INRIA Futurs & LSV, UMR 8643 CNRS/ENS-Cachan
† LORIA & INRIA Lorraine, UMR 7503
‡ LORIA & Univ. Nancy 2, UMR 7503

Unité de recherche INRIA Futurs
Parc Club Orsay Université, ZAC des Vignes,

Automates d'arbres à contraintes égalitaires modulo des théories équationnelles

Résumé : Ce document présente de nouvelles classes d'automates d'arbres qui sont des combinaisons d'automates à contraintes égalitaires et d'automates modulo des théories équationnelles. Ces automates sont définis par l'extension de la représentation classique par clauses de Horn avec des équations et des règles de réécriture. Nous montrons que le problème de l'appartenance à un langage d'automate donné d'une instance close d'un terme donné (généralisant le problème du vide) est décidable, à l'aide de techniques de saturation utilisant des stratégies de paramodulation appropriées. Nous décrivons bièvement une application de ces classes d'automates d'arbres au problème de l'accessibilité dans un fragment du pi-calcul appliqué.

Mots-clés : Déduction automatique, automates d'arbres, paramodulation basique, splitting, vérification de systèmes infinis.

1 Introduction

Tree automata have appeared to be a well suited formalism for the verification of reachability properties of infinite state systems, see *e.g.* [LS02, AL00, GK00, NRNS02, Mon03, KW04]. Roughly, in this setting, the states of the system are represented as ground terms (terms without variables) on a first order signature, a term rewriting system (TRS) is used to describe some transitions and the set of reachable states, or an approximation, is characterized by a tree automaton. There are two important issues with such approaches: 1. the preservation of regularity of term languages under term rewriting, in order to apply known tree automata algorithms to the sets of reachable states, and 2. the non linear variables (variables with multiple occurrences) in the rewrite rules, which impose in general some over-approximations of the rewrite relation. It is well known indeed that the language of ground instances of *e.g.* $f(x, x)$ is not regular (this is classically shown with a *pumping lemma*). The above two issues have in general been addressed separately so far.

There have been a lot of results about regularity preservation under rewriting during the last decades (see *e.g.* [STFK02], Section 2.3 for a summary of some important results). Some authors (*e.g.* [Ver03, OT02]) have also investigated the problem of emptiness decision for tree automata modulo ad-hoc equational theories, *e.g.* AC, ACU, ...

Tree automata with constraints have been proposed earlier in order to deal with non-linear rewrite systems (see [CDG+97]). They are an extension of classical tree recognizers where syntactic equality and disequality tests between subterms are performed during the automata computations. It is shown in [Mon81] that emptiness of the recognized language is undecidable without restriction. Two remarkable subclasses with decidable emptiness problem are the tree automata with equality and disequality constraints restricted to brother positions of [BT92] and the *reduction automata* of [DCC95]. This second class permits one to characterize in particular languages of terms reducible by non linear rewrite systems. Emptiness decidability is ensured with a restriction on the transitions for bounding the number of equality tests that can be performed along a computation path.

Following [FSVY91], it is classical to represent tree automata by Horn clause sets. In this setting, a recognized language is defined as a least Herbrand model. This representation permits to use classical first-order theorem proving techniques in order to establish decision results for tree automata [Ver03, Gou05, LS, JMW98]. The last reference in particular is concerned with decision problems for tree automata with equality constraints between brother positions (à la [BT92]) modulo shallow equational theories.

In this paper, we show how techniques of basic ordered paramodulation with selection and a variant of splitting without backtracking solve some decision problems on tree automata languages transformed by rewriting. More precisely, we show the the so called *Generalized Membership Problem*, GMP (whether there exists a ground instance of a given term in a given language) is decidable by saturation with a standard calculus presented in Section 3. Note that GMP generalizes the emptiness problem. Both classes of standard tree automata (TA) and tree automata with equality constraints generalizing those of [DCC95], where the equality tests are presented by arbitrary equations (TAD), are studied in these settings, as well as their respective generalisation modulo an equational theory $\mathcal{E}$ presented as a convergent term rewriting system (monadic TRS in the case of TA and restricted collapsing TRS in the case of TAD). The decision results are presented as follows in the paper:

	$\mathcal{E} = \emptyset$	$\mathcal{E}$
TA	Section 4	Section 6
TAD	Section 5	Section 7

The last result (lower right corner of the table) is to our knowledge one of the first decision results (after [JMW98]) concerning tree automata with equality constraints modulo equational theories. We show that emptiness is undecidable for TA extended with non-linear facts, even with only one state. Unlike stated in [DCC95, CDG+97], it appears also that this problem is undecidable for non-deterministic reduction automata (see Section 5). Therefore, we have introduced for the definition of TAD a refinement on the restriction for the automata of [DCC95] in order to make GMP decidable. The idea is roughly to bound the number of equality tests that can be performed along a whole computation (and not only along each computation path).

The representation of constrained automata as Horn clauses permits us to use state of the art first-order theorem proving techniques to provide an effective (implementable) decision procedure for GMP (hence emptiness), instead of the complicated pumping lemmas used so far which hardly lead to effective algorithms. A key-ingredient for the termination of our saturation-based decision procedure was the application of recently proposed *splitting* rules.

The class of automata of Section 7 permits a sharper modeling of verification problems (avoiding approximation as it is often required with more standard tree automata), thanks to the equational constraints. We show

in Section 8 how our decision results can be applied for the verification of reachability problems in a fragment of the applied pi calculus.

Related work. A comparison with the reduction automata of [DCC95] is detailed in Sections 5 and 7.

The closely related works [NRNS02, Gou05] propose a different extension H_1 of standard TA defined as Horn clause sets for which satisfiability is decidable. In the version [Gou05] of H_1 Horn clauses have a head whose argument is at most of height one and linear (without duplicated variables), or are purely negative (goals). None of the classes TAD and H_1 contains the other. However, H_1 becomes undecidable when allowing variable duplication in the heads. Our TAD class allows this under the previously mentioned restrictions.

2 Preliminaries

Term algebra. Let $\mathcal{F}$ be a signature of function symbols with arity, denoted by lowercase letters f, $g\ldots$ and let $\mathcal{X}$ be an infinite set of variables. The term algebra is denoted $\mathcal{T}(\mathcal{F},\mathcal{X})$, and $\mathcal{T}(\mathcal{F})$ for ground terms. A term is called *linear* if every variable occurs at most once in it and *sublinear* if all its strict subterms are linear. We note $vars(t)$ the set of variables occurring in a term $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$. A substitution σ is a mapping from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F},\mathcal{X})$ such that $\{x|\sigma(x) \neq x\}$, the *support* of σ, is a finite set. The application of a substitution σ to a term t is denoted by $t\sigma$ and is equal to the term t where all variables x have been replaced by the term $\sigma(x)$. A substitution σ is grounding for t if $t\sigma \in \mathcal{T}(\mathcal{F})$. The *positions* $Pos(t)$ in a term t are represented as sequence of positive integers (Λ, the empty sequence, denotes the root position). A subterm of t at position p is denoted $t|_p$, and the replacement in t of the subterm at position p by u denoted $t[u]_p$.

Rewriting. A *term rewriting system* (TRS) is a finite set of rewrite rules $\ell \to r$, where $\ell \in \mathcal{T}(\mathcal{F},\mathcal{X})$ and $r \in \mathcal{T}(\mathcal{F}, vars(\ell))$. A term $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ rewrites to s by a TRS $\mathcal{R}$ if there is a rewrite rule $\ell \to r \in \mathcal{R}$, a position p of t and a substitution σ such that $t|_p = \ell\sigma$ and $s = t[r\sigma]_p$. A TRS is terminating if there is no infinite chain of terms s_i, $i \in \mathbb{N}$, such that s_i rewrites to s_{i+1}. A term t' is a *normal form* of a term t for a TRS $\mathcal{R}$ if it is the result of a sequence of rewriting on t and cannot be rewritten to another term. A TRS $\mathcal{R}$ is *confluent* if every term has a unique normal form.

Clauses. Let $\mathcal{P}$ be a finite set of predicate symbols which contains an equality predicate $=$. The other predicate symbols are denoted by uppercase letter P, Q,... and are assumed unary. We shall use later a partition $\mathcal{P} \setminus \{=\} = \mathcal{P}_0 \uplus \mathcal{P}_1$. Let $\mathcal{Q}$ be a finite set of nullary predicate symbols disjoint from $\mathcal{P}$ and that we call *splitting predicates*, denoted by lowercase letters $q\ldots$ Constrained Horn clauses are constrained disjunctions of literals denoted $\Gamma \Rightarrow H \ [\![\theta]\!]$ where Γ is a set of negative literals called *antecedents*, H is a positive literal called *head* of the clause and the constraint θ is a set of equations between terms of $\mathcal{T}(\mathcal{F},\mathcal{X})$. A clause with a splitting literal as head or with no head at all is called a *goal*. The constraint is omitted when θ is empty. For sake of notation, we shall sometimes make no distinction between the constraint and its most general solution (when it exists). When θ is satisfiable, we call *expansion* of the above clause the unconstrained clause $\Gamma\theta \Rightarrow H\theta$.

Atoms of the form $P(s)$, resp. q, where $P \in \mathcal{P}$ and $s \in \mathcal{T}(\mathcal{F},\mathcal{X})$, resp. $q \in \mathcal{Q}$, are represented for uniformity as equations $P(s) = true$, resp. $q = true$, where $true$ is a distinguished function symbol (in $\mathcal{F}$). An atom of the latter form is called *non-equational* atom and can be denoted simply $P(s)$, resp. q. We assume in the following that predicate symbols can only occur at the root of the terms that we consider.

Orderings. We assume given a *precedence* ordering $\succeq$ on $\mathcal{F} \cup \mathcal{P} \cup \mathcal{Q}$, and denote $\sim$ the relation $\preceq \cap \succeq$ and $\succ$ the relation $\succeq \setminus \sim$. We assume that $\succ$ is total on $\mathcal{P}_1$ and moreover that for all predicates $P_0, P_0' \in \mathcal{P}_0$, $P_1 \in \mathcal{P}_1$, $q \in \mathcal{Q}$ and every function symbol $f \in \mathcal{F}$, $P_0 \sim P_0'$ and $P_1 \succ P_0 \succ q \succ f$. We assume the symbol $true$ to be the minimal one. Assume that $\mathcal{P}_1 = \{P_1, \ldots, P_n\}$ with $P_1 \succ \ldots \succ P_n$. We call i the *index* of P_i, denoted $ind(P_i)$, and let $ind(Q) = 0$ for all $Q \in \mathcal{P}_0$. We shall also use the constant $\infty = \max(ind(P)|P \in \mathcal{P}) + 1$, which is bigger than the index of every predicate in $\mathcal{P}_1$.

A *reduction ordering* $>$ is a well-founded ordering on $\mathcal{T}(\mathcal{F} \cup \mathcal{P} \cup \mathcal{Q}, \mathcal{X})$ stable under substitutions and such that for all $g \in \mathcal{F} \cup \mathcal{P} \cup \mathcal{Q}$, for all $s,t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ we have $s > t$ implies $g(\ldots,s,\ldots) > g(\ldots,t,\ldots)$. The *multiset* extension $>^{mul}$ of an ordering $>$ is defined as the smallest ordering relation on multisets such that $M \cup \{t\} >^{mul} M \cup \{s_1,\ldots,s_n\}$ whenever $t > s_i$ for all $i \leq n$. The *lexicographic* extension $(>_1,\ldots,>_n)^{lex}$ of n orderings to n-tuples is defined as $(s_1,s_2,\ldots,s_n)(>_1,\ldots,>_n)^{lex}(t_1,t_2,\ldots,t_n)$ if $s_1 = t_1$,..., $s_{k-1} = t_{k-1}$, and $s_k >_k t_k$ for some $k \in 1\ldots n$.

We can define a reduction ordering on $\mathcal{T}(\mathcal{F} \cup \mathcal{P} \cup \mathcal{Q}, \mathcal{X})$ total on ground terms by extending the precedence $\succ$ to a *lexicographic path ordering* [DJ90] denoted $\succ_{lpo}$ with: $s = f(s_1, s_2, \ldots, s_m) \succ_{lpo} g(t_1, t_2, \ldots, t_n) = t$ iff

1. $f \succ g$ and $s \succ_{lpo} t_i$ for all $1 \leq i \leq n$; or
2. $f \sim g$ and, for some j, we have $(s_1, \ldots, s_{j-1}) = (t_1, \ldots, t_{j-1})$, $s_j \succ_{lpo} t_j$ and $s \succ_{lpo} t_k$, for all k with $j < k \leq n$; or
3. $s_j \succ_{lpo} t$, for some j with $1 \leq j \leq m$.

Then as in [BGLS95] we identify a positive literal $s = t$ with the multiset $\{\{s\}, \{t\}\}$, and a negative literal $s \neq t$ with the multiset $\{\{s, t\}\}$. Then we extend the ordering $\succ_{lpo}$ (resp. $\succ_{lpo}$) to literals by taking the twofold multiset ordering $((\succ_{lpo})^{mul})^{mul}$ (resp. $((\succ_{lpo})^{mul})^{mul}$).

Tree Automata. We consider tree automata defined à la Frühwirth et al [FSVY91] as finite sets of Horn clauses on $\mathcal{P}$ and $\mathcal{F}$ with equality. Every non-equational predicate symbol occurring in a given tree automaton $\mathcal{A}$ is called a *state* of $\mathcal{A}$. Given a tree automaton $\mathcal{A}$ and a state $Q \in \mathcal{P}$ of $\mathcal{A}$, the language of $\mathcal{A}$ in Q, denoted $L(\mathcal{A}, Q)$, is the set of terms $t \in \mathcal{T}(\mathcal{F})$ such that $Q(t)$ is a logical consequence of $\mathcal{A}$.

General Membership Problem (GMP). We focus on one decision problem, GMP, which generalizes many important problems concerning tree automata (in particular membership and emptiness decision). This problem has been shown decidable in [Tis00] for standard tree automata.
INSTANCE: a tree automaton $\mathcal{A}$, a state Q of $\mathcal{A}$ and a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$,
QUESTION: does there exist a substitution σ grounding for t such that $t\sigma \in L(\mathcal{A}, Q)$?
In particular, when t is a ground term, this problem is equivalent to a *membership* problem for $\mathcal{A}$: $t \in L(\mathcal{A}, Q)$? When t is a variable, it is equivalent to a *non-emptiness* problem for $\mathcal{A}$: $L(\mathcal{A}, Q) \neq \emptyset$?

Lemma 1 *GMP is satisfied by $\mathcal{A}$, Q and t iff $\mathcal{A} \cup \{Q(t) \Rightarrow\}$ is inconsistent.*

3 Basic Ordered Paramodulation with Selection

We shall establish the decidability of GMP for several classes of tree automata (with equations), using techniques of saturation under paramodulation, based on Lemma 1 and the calculus described in this section.

Basic Ordered Paramodulation with Selection. The following set of inference rules, parametrized by a reduction ordering $\succ$, which we assume total on ground terms, and a function of selection of negative literals[1], forms a sound and refutationally complete (*i.e.* for every unsatisfiable set of clauses the inference system will generate, with a fair strategy, the empty clause) calculus for Horn clauses called *basic ordered paramodulation with selection* [BGLS95, NR01].

$$\frac{\Gamma \Rightarrow \ell = r \; [\![\theta]\!] \quad \Gamma' \Rightarrow u[\ell']_p = v \; [\![\theta']\!]}{\Gamma, \Gamma' \Rightarrow u[x]_p = v \; [\![\theta, \theta', \ell' = \ell, x = r]\!]} \; \mathsf{RP}$$

if x is fresh, (i) $\ell' \notin \mathcal{X}$, (ii) no literal is selected in Γ and Γ', (iii) $\ell\sigma \not\preceq r\sigma$ and $\ell\sigma = r\sigma$ is strictly maximal in $\Gamma\sigma, \ell\sigma = r\sigma$, (iv) $u\sigma = v\sigma$ is maximal in $\Gamma'\sigma, u\sigma = v\sigma$, where σ is the mgu of $\theta, \theta', \ell' = \ell, x = r$.

$$\frac{\Gamma \Rightarrow \ell = r \; [\![\theta]\!] \quad \Gamma', u[\ell']_p = v \Rightarrow A \; [\![\theta']\!]}{\Gamma, \Gamma', u[x]_p = v \Rightarrow A \; [\![\theta, \theta', \ell' = \ell, x = r]\!]} \; \mathsf{LP}$$

if x is fresh, (i) $\ell' \notin \mathcal{X}$, (ii) no literal is selected in Γ, (iii) $\ell\sigma \not\preceq r\sigma$ and $\ell\sigma = r\sigma$ is strictly maximal in $\Gamma\sigma, \ell\sigma = r\sigma$, (iv) $u = v$ is selected or (v) $u\sigma = v\sigma$ is maximal in $\Gamma'\sigma, u\sigma = v\sigma, A\sigma$, where σ is the mgu of $\theta, \theta', \ell' = \ell, x = r$.

$$\frac{\Gamma, s = t \Rightarrow A \; [\![\theta]\!]}{\Gamma \Rightarrow A \; [\![\theta, s = t]\!]} \; \mathsf{Eq}$$

if (i) $s = t$ is selected or (ii) $s\sigma \not\preceq t\sigma$ and $s\sigma = t\sigma$ is maximal in $\Gamma\sigma, s\sigma = t\sigma, A\sigma$, where σ is the mgu of $\theta, s = t$.

Concerning RP and LP, we shall talk of paramodulation of the first clause (called first *premise*) into the second clause (second *premise*). The clause returned by the above inferences is called *conclusion*. If after every step the constraints are eagerly propagated in the clauses (*i.e.* each clause is expanded) the calculus is called *ordered paramodulation with selection*.

[1] We shall sometimes underline literals to indicate that they are selected.

Resolution. The application of LP into non-equational atoms followed by Eq is called *basic resolution.*

$$\frac{\Gamma \Rightarrow P(\ell) = \mathit{true}\ [\![\theta]\!] \quad \Gamma', P(\ell') = \mathit{true} \Rightarrow A\ [\![\theta']\!]}{\Gamma, \Gamma' \Rightarrow A\ [\![\theta, \theta', \ell' = \ell]\!]}\ \mathsf{R}$$

When the non-basic version of LP and Eq are used, this inference is simply called *ordered resolution.*
Note that when the unconstrained part of a clause only contains variables (no function symbols), only the resolution rule applies into this clause, and the clause obtained also contains only variables (*i.e.* every application of LP is performed at the root position of an atom). Therefore, for the sake of presentation, we shall eagerly apply the constraint when describing the application of R in this case. The application of RP to clauses whose heads are non-equational returns a tautology, and hence this case will be ignored in the following proofs.

Deletion of redundant clauses. We assume that the deletion of tautologies and subsumed clauses (these notions are considered after clause expansion) and the simplification under rewriting by orientable positive equational clauses are applied as in [BGLS95].

Splitting. We shall use *ε-splitting* [Gou05], a variant of *splitting without backtracking* [RV01].

$$\frac{B, \Gamma' \Rightarrow H[\![\theta]\!]}{\Gamma \Rightarrow q_B[\![\theta]\!] \quad q_B, \Gamma' \Rightarrow H[\![\theta]\!]}\ \varepsilon\mathsf{split}$$

where the literals of $\Gamma' \cup H$ are not equational, $B\theta$ is an *ε-block*, *i.e.* a set of literals of the form $Q_1(x), \ldots, Q_n(x)$, with $Q_1, \ldots, Q_n \in \mathcal{P}$, x is a variable which does not occur in Γ' and H, and where $q_B \in \mathcal{Q}$ is uniquely associated to B, modulo variable renaming.

Note that the above splitting rule *replaces* a clause by two split clauses. Using this rule eagerly (as soon as possible) preserves correctness and completeness of the calculus. Indeed, since every splitting predicate q_B is smaller than any predicate of $\mathcal{P}$, the original clause is redundant (wrt the general redundancy criterion of [BGLS95]) because its reduced instances are implied by smaller reduced instances of the split clauses. Another important point is that the number of splitting literals that can be introduced is bounded. We will assume that the set $\mathcal{Q}$ is large enough to cover all ε-blocks.

4 Standard Tree Automata

The transitions of standard tree automata are classically encoded into Horn clauses of the following form:

$$Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big) \qquad (\mathsf{s})$$

where $n \geq 0$ (when $n = 0$, by convention, the set of antecedents of the clause is empty), x_1,...,x_n are distinct variables and $Q_1, \ldots, Q_n, Q \in \mathcal{P}_0$.

Definition 1 *A standard bottom-up tree automaton (TA) is a finite set of clauses of type (s).*

The language of a TA is called a *regular* language. The emptiness and membership problems can be solved in deterministic time, respectively linear and quadratic. GMP for a linear term can be decided by a procedure of the same quadratic time complexity. For a non-linear term, the problem becomes EXPTIME-complete. We present below a slight variation of a DEXPTIME procedure of [Gou05] in our framework in order to introduce the principles of the proofs in the next sections.

Proposition 1 ([Gou05]) *GMP for TA can be solved using ordered resolution with selection and ε-splitting.*

*Proof.*Let sel_1 be a selection function which selects in a Horn clause $\Gamma \Rightarrow H[\![\theta]\!]$:

- every splitting negative literal, if any,
- otherwise every non-equational literal $Q(t)$ of Γ such that $t\theta$ is not a variable.

We assume given a TA $\mathcal{A}$ and a goal clause $P(t) \Rightarrow$ and consider ordered resolution wrt $\succ_{lpo}$ and the above selection function sel_1 with eager application of the εsplit rule of Section 3. We assume wlog that $P \in \mathcal{P}_0$, otherwise the problem is trivial by definition of (s). We show that all the clauses generated have one of the following form (gs), for goal-subterm, or (gf), for goal-flat.

$$\underline{q_1}, \ldots, \underline{q_k}, P_1(s_1), \ldots, P_m(s_m) \Rightarrow [q] \qquad \text{(gs)}$$

where $m, k \geq 0$, $s_1, \ldots, s_m \in st(t)$. $P_1, \ldots, P_m \in \mathcal{P}_0$, and $q_1, \ldots, q_k, q$ are splitting literals (the q in the head is optional, as indicated by the square brackets).

$$P_1(y_{i_1}), \ldots, P_k(y_{i_k}), \underline{P'_1(f(y_1, \ldots, y_n))}, \ldots, \underline{P'_m(f(y_1, \ldots, y_n))} \Rightarrow [q] \qquad \text{(gf)}$$

where $k, m \geq 0$, $i_1, \ldots, i_k \leq n$, $y_1, \ldots, y_n$ are distinct variables, $P_1, \ldots, P_k, P'_1, \ldots .P'_m \in \mathcal{P}_0$, and q is a splitting literal (it is optional in the clause). Let us consider the particular subtype (sp) of (gf) of positive clauses with a splitting literal as head (i.e. (gf) with $k = m = 0$):

$$\Rightarrow [q] \qquad \text{(sp)}$$

where $q \in \mathcal{Q}$ (note that this type contains the empty clause).

The initial goal $P(t) \Rightarrow$ belongs to the type (gs) and also belongs to (gf) if t is a variable. The different cases of resolution steps between clauses of type (s), (gs) and (gf) are listed in Appendix A. Since the number of clauses of type (gs) and (gf) is exponential, the saturation terminates and GMP is solvable in deterministic exponential time. □

Undecidable extension. Let us call *fact* a Horn clause $\Rightarrow H$ with no antecedents at all. We define a clause to be of type (s_+) if it is of type (s) or a fact. Note that we allow non-linear variables in facts. We can show that GMP for this slight extension of TA is undecidable [2]:

Proposition 2 *GMP for sets of clauses of type (s_+) is undecidable (even with one predicate).*

Proof. We reduce the halting problem of 2 counter machines to GMP for (s_+). Let us consider a deterministic 2-counter machine such that q_0 is the initial state and q_f the final one (from where no transition is possible). A configuration of the machine can be represented by a term $q(s^n(0), s^m(0))$ where q is the state, and n (resp. m) the value of the first (resp. second) counter. We encode every transition $q(s,t) \rightarrow q'(s',t')$ of the machine by a fact. We will need a predicate symbol Q, binary functions g, h, k and a constant symbol c:

MACHINE INSTRUCTION	CLAUSAL REPRESENTATION
q : ADD 1 TO COUNTER 1; GOTO q'	$\Rightarrow Q(g(q(x,y), h(g(q'(s(x),y),u),u)))$
q : IF COUNTER 1 $\neq 0$ DEC 1; GOTO q'	$\Rightarrow Q(g(q(s(x),y), h(g(q'(x,y),u),u)))$
q : IF COUNTER 1 $= 0$; GOTO q'	$\Rightarrow Q(g(q(0,y), h(g(q'(0,y),u),u)))$

See also Figure 3 for a representation of these three examples. We add two facts to detect the halting state: $\Rightarrow Q(g(q_\mathsf{f}(x,y),c))$ and $\Rightarrow Q(c)$. We introduce two auxiliary (s) clauses:

$$Q(x_1), Q(x_2) \Rightarrow Q(h(x_1, x_2)) \qquad \text{(h)}$$

$$Q(x_1), Q(x_2) \Rightarrow Q(k(x_1, x_2)) \qquad \text{(k)}$$

and finally we introduce a goal clause: $Q(k(y, g(q_0(0,0), y))) \Rightarrow$. We shall employ a resolution strategy with selection function sel_1. A first resolution step of (k) into the initial goal clause generates: $Q(y), \underline{Q(g(q_0(0,0),y))} \Rightarrow$. Then a resolution with a fact encoding a transition $q_0(0,0) \rightarrow st$, for some term st, generates: $\underline{Q(h(g(st,u),u))} \Rightarrow$, which is resolved by (h) to produce $Q(u), \underline{Q(g(st,u))} \Rightarrow$. This process can be iterated. The halting state can be reached iff some goal clause is derived that can be resolved with $\Rightarrow Q(g(q_f(x,y),c))$, producing $Q(c) \Rightarrow$ which in turn generates the empty clause with $\Rightarrow Q(c)$. Hence the set of clauses encoding the machine is unsatisfiable iff the machine halts. □

[2] GMP with linear facts can be shown decidable.

5 Tree Automata with Syntactic Equational Constraints

Reduction Automata. The original reduction automata of [DCC95] can be defined as a set of constrained Horn clauses of the following form:

$$Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q(f(x_1, \ldots, x_n))[\![c]\!] \qquad (\mathsf{red})$$

where $n > 0$, $x_1, \ldots, x_n$ are distinct variables, c is a conjunction of constraints of the form $x_i|_p = x_{i'}|_{p'}$ (equality constraint) or $x_i|_p \neq x_{i'}|_{p'}$ (equality constraint) for some positions p and p' (sequences of integers), Q is maximal in $\{Q, Q_1, \ldots, Q_n\}$ (here, we do not assume that the ordering on predicates is total) and it is moreover *strictly* maximal if c contains at least one equality constraint. An equality constraint as above (resp. disequality constraint) is satisfied by every two ground terms $t, t' \in \mathcal{T}(\mathcal{F})$ such that $p \in Pos(t)$, $p' \in Pos(t')$ and $t|_p = t'|_{p'}$ (resp. $p \in Pos(t)$, $p' \in Pos(t')$ and $t|_p \neq t'|_{p'}$).

It appears that the emptiness problem is undecidable for non-deterministic reduction automata, contradicting a claim in [DCC95, CDG+97].

Proposition 3 *The emptiness problem is undecidable for non-deterministic reduction automata.*

The proof, a variation of the above proof of Proposition 2, is given in Appendix B.

TAD. We propose here the definition of a new class of tree automata where the constraints are generalized to equations between arbitrary terms and where the transitions comply to stronger (compared to [DCC95]) ordering conditions, based on the ordering $\succ$ on states, in order to obtain a decidable GMP. We call below *test predicates*[3] the elements of $\mathcal{P}_1$. The constrained transitions of our automata have the following form:

$$Q_1(x_1), \ldots, Q_n(x_n), u_1 = v_1, \ldots, u_k = v_k \Rightarrow Q^*(x) \qquad (\mathsf{d})$$

where $n, k \geq 0$, $x_1, \ldots, x_n, x$ are distinct variables, $u_1, v_1, \ldots, u_k, v_k \in \mathcal{T}\big(\mathcal{F}, \{x_1, \ldots, x_n, x\}\big)$, $Q_1, \ldots, Q_n, Q \in \mathcal{P}$, Q^* is a test predicate, and for all $i \leq n$, if Q_i is a test predicate then $Q^* \succ Q_i$.

The unconstrained transitions are restricted to clauses of type (red) which contain no more test predicates symbols in their antecedents that in their heads.

$$Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big) \qquad (\mathsf{t})$$

where $n > 0$, $x_1, \ldots, x_n$ are distinct variables, and either $Q_1, \ldots, Q_n, Q \in \mathcal{P}_0$ or Q is a test predicate and at most one of $Q_1, \ldots, Q_n$ is equal to Q, and the others belong to $\mathcal{P}_0$.

Definition 2 *A tree automaton with equational constraints or TAD is a finite set of clauses of type (t) or (d).*

Note that every TA is a particular case of TAD (without test predicates).

Proposition 4 *GMP for TAD can be solved using ordered paramodulation with selection and ε-splitting.*

Proof. Let sel_2 be a selection function which selects in a Horn clause $C[\![\theta]\!]$:

- every splitting negative literal, if any,
- otherwise, the equational negative literals of C, if C contains any,
- otherwise, every (non-equational and non-splitting) negative literal $Q(t)$ of C such that $t\theta$ is not a variable (if any).

We assume given a TAD $\mathcal{A}$ and a goal clause $P(t) \Rightarrow$ and we are going to show the termination of the saturation of $\mathcal{A} \cup \{P(t) \Rightarrow\}$ under ordered paramodulation wrt the ordering $\succ_{lpo}$ and the selection function sel_2, with eager ε-splitting.

Note first that with our selection strategy, the clauses of $\mathcal{A}$ of type (d) containing equations can only be involved in an equality resolution (Eq). Every such application solves (and remove) an equation (selected by

[3] and we shall sometimes mark a predicate Q with an asterisk like in Q^* to indicate that it is a test predicate.

sel_2), hence saturation of the clauses of type (d) under (Eq) terminates. Let us call (d_+) the type of clauses obtained this way (and with ε-splitting) from clauses of (d) and which contain no more equations.

$$q_1, \ldots, q_k, Q_1(s_1), \ldots, Q_n(s_n) \Rightarrow Q^*(s) \qquad (\mathsf{d}_+)$$

where $k, n \geq 0$, $q_1, \ldots, q_k \in \mathcal{Q}$, $Q_1, \ldots, Q_n, Q^* \in \mathcal{P}$, $s_1, \ldots, s_n, s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, Q^* is a test predicate, and for all $i \leq n$, if Q_i is a test predicate then $Q^* \succ Q_i$.

It order to analyse the type of clauses obtained by resolution, we shall consider the clause types (gf) and (sp) defined in the proof of Proposition 1, and the type (g_+) of arbitrary goals:

$$q_1, \ldots, q_k, P_1(s_1), \ldots, P_m(s_m) \Rightarrow [\, q \,] \qquad (\mathsf{g}_+)$$

where $k, m \geq 0$, $P_1, \ldots, P_m \in \mathcal{P}$, $q_1, \ldots, q_k, q \in \mathcal{Q}$ and $s_1, \ldots, s_m \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

The head is optional in the clause, as indicated by the brackets, and can only be a splitting literal (hence we abusively call such a clause a goal). Note also that the initial goal $P(t) \Rightarrow$ has type (g_+). The measure of a clause $C = \Gamma, \Xi \Rightarrow H[\![\theta]\!]$, where Γ is a multiset of non-equational atoms and Ξ is a multiset of equations, is the tuple made of the following components:

- $m_1(C) = ind(Q)$ (see page 4 for the definition of ind) if $H = Q(t)$ with $Q \in \mathcal{P}$, or $m_1(C) = \infty$ if C has type (sp), $m_1(C) = 0$ if C is a goal not of type (sp), *i.e.* if Γ is not empty and H is a splitting literal or there is no H,
- $m_2(C)$ is the number of equations in Ξ,
- $m_4(C)$ is the multiset of test predicate symbols occurring in Γ,
- $m_6(C)$ is the multiset of the negative non-equational literals of $\Gamma\theta$.

The strict ordering $\gg$ on measures, extended as expected to clauses, is defined as the lexicographic extension $\left(>, >, \succ^{mul}, \succ^{mul}_{lpo}\right)^{lex}$, where $>$ denotes the ordering on natural numbers. The following Fact is proved by a case analysis summarized in Figure 1 and detailed in Appendix C.

Fact 1 Starting with $\mathcal{A} \cup \{P(t) \Rightarrow\}$, every step of ordered paramodulation with selection by sel_2 and ε-splitting returns either a clause smaller than all its premises (wrt $\gg$) or a clause of type (gf).

Fact 1 permits us show that ordered paramodulation with selection and ε-splitting saturates $\mathcal{A}$ and the goal, hence to conclude the proof of the proposition. Indeed, the number of clauses of type (gf) is finite up to variable renaming, hence an infinite deduction path would contain an infinite decreasing chain, wrt $\gg$, whereas this order is well-founded. □

6 Tree Automata Modulo Monadic Theories

There have been many works to identify some classes of rewrite systems preserving the regularity of sets of terms, like for instance ground TRS, right-linear monadic TRS, linear semi-monadic TRS... (see [STFK02], Section 2.3 for a summary of some recent results). These results often rely on a procedure of *completion* of TA w.r.t. some TRS, which adds new TA transitions without adding new states. As observed in [JMW98], such a TA completion can be simulated by saturation under paramodulation. The next results show that this method is effective (*i.e.* terminates) in the case of monadic theories.

Definition 3 *A rewrite rule $\ell \to r$ is called* sublinear *if ℓ is sublinear,* collapsing *if r is either a ground term or a variable, and* monadic *if r is either a variable occurring in ℓ or a term $g(z_1, \ldots, z_k)$ for some $g \in \mathcal{F}$, $k \geq 0$ and some distinct variables $z_1, \ldots, z_k$ occurring in ℓ.*

Example 1 The rule $\mathsf{dec}(\mathsf{enc}(x, y), y) \to x$ for decryption in symmetric key cryptography is sublinear and collapsing. The function symbol enc and dec stand for encryption and decryption and the variables x and y correspond respectively to the encrypted plaintext and the encryption key.

inf.	1^{st} pr.	2^{nd} pr.	cl	1^{st} premise				2^{nd} premise			
				m_1	m_2	m_4	m_6	m_1	m_2	m_4	m_6
Eq	d		d/d_+	$=$	$>$	_	_				
R	_	t	$\nearrow$								
R	_	d_+	$\nearrow$								
R	t	$\mathsf{g}_+{}^{(1)}$	g_+	$>$	_	_	_	$=$	$=$	$\geq$	$>$
R	t	$\mathsf{g}_+{}^{(2)}$	gf								
R	t	g_+	sp								
R	t	gf	gf								
R	d_+	g_+	g_+	$>$	_	_	_	$=$	$=$	$>$	_
R	d_+	g_+	sp								
R	sp	$\mathsf{d}_+{}^{(3)}$	d_+	$>$	_	_	_	$=$	$=$	$=$	$>$
R	sp	$\mathsf{g}_+{}^{(3)}$	g_+	$>$	_	_	_	$=$	$=$	$=$	$>$
R	sp	$\mathsf{g}_+{}^{(3)}$	sp								
εsplit	d_+		gf								
εsplit	d_+		d_+	$=$	$=$	$\geq$	$>$				
εsplit	g_+		gf								
εsplit	g_+		g_+	$=$	$=$	$\geq$	$>$				

(1) no negative splitting literals and at least one literal selected
(2) no literal selected (3) at least one negative splitting literal (selected)

Figure 1: Case analysis in the proof of Proposition 4. $>$ (resp. $\geq$, $=$) means that the measure's component for the premise is strictly greater than (resp. greater or equal to, equal to) the conclusion.

We call *equational theory* a set of positive clauses of the form:

$$\Rightarrow \ell = r \qquad (\mathsf{eq})$$

An equational theory $\mathcal{E}$ is called $\succ$-*convergent* if for each clause of $\mathcal{E}$, the equation $\ell = r$ is orientable by $\succ_{lpo}$, *i.e.* $\ell \succ_{lpo} r$, and the rewrite system $\mathcal{R} = \{\ell \to r \mid \Rightarrow \ell = r \in \mathcal{E} \text{ and } \ell \succ_{lpo} r\}$ is confluent. Moreover, the theory $\mathcal{E}$ is called sublinear (resp. collapsing, monadic) if all the rules of $\mathcal{R}$ are sublinear (resp. collapsing, monadic).

Definition 4 *A tree automaton modulo an equational theory (TAE) is the union of an equational theory and of a finite set of clauses of type (*s*).*

Proposition 5 *GMP for TAE modulo a $\succ$-convergent monadic equational theory can be solved using basic ordered paramodulation with selection and non ground splitting.*

Proof. Let $\mathcal{A}$ be a TAE modulo a $\succ$-convergent monadic equational theory (presented by a convergent TRS $\mathcal{R}$) and let $P(t) \Rightarrow$ be a goal clause. We assume wlog that $P \in \mathcal{P}_0$ (otherwise the problem is trivial). We consider basic ordered paramodulation wrt the ordering $\succ_{lpo}$ and the selection function sel_1 defined in the proof of Proposition 1 and with eager ε-splitting.

With the equational theory, RP can now be applied into a clause (s):

$$\frac{\Rightarrow f(\ell_1, \ldots, \ell_n) = r \quad Q_1(x_1), \ldots, Q_1(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big)}{Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q(y)\ [\![x_1 = \ell_1, \ldots, x_n = \ell_n, y = r]\!]}\ \mathsf{RP}$$

Also, LP with an equational clause (eq) is possible into the initial goal clause $P(t) \Rightarrow$. We introduce the following type (I) to characterize the (expansions of) clauses obtained this way:

$$\underline{q_1}, \ldots, \underline{q_k}, Q_1(s_1), \ldots, Q_n(s_n) \Rightarrow [\,H\,] \qquad (\mathsf{I})$$

where $k, n \geq 0$, every s_i, $i \leq n$ is either a subterm of a left hand side of a rule of $\mathcal{R}$ or a term obtained from t by basic narrowing with $\mathcal{R}$ (see Appendix D for a longer description), $q_1, \ldots, q_k \in \mathcal{Q}$, $Q_1, \ldots, Q_n \in \mathcal{P}_0$, and H is $Q(r)$ for some $Q \in \mathcal{P}_0$ and r is either a variable $x \in vars(s_1, \ldots, s_n)$ a flat term $g(z_1, \ldots, z_m)$ ($k \geq 0$) whose variables $z_1, \ldots, z_m$ belong to $vars(s_1, \ldots, s_n)$ and are pairwise distinct, or H is a splitting literal or else there is no H. Note that (sp) is a subcase of (I) and that the initial goal $P(t) \Rightarrow$ also belongs to this type.

We show in Appendix D that all the clauses obtained by basic ordered paramodulation starting from $\mathcal{A} \cup \{Q(t) \Rightarrow\}$ are of type (l) or of a type (f) which generalizes (gf) (proof of Proposition 1) allowing a head $Q(r)$ as in the definition of (l) above. Since the number of clauses of type (l) and (f) is finite, this shows that the saturation of $\mathcal{A} \cup \{P(t) \Rightarrow\}$ under basic ordered paramodulation terminates. □

7 Tree Automata with Equational Constraints Modulo

It is shown in [JMW98] that the class of languages of terms recognized by tree automata of [BT92] with equality constraints between brother positions are not closed under rewriting with shallow theories (rewrite systems whose left and right members of rules have depth 1). The reason is that these tree automata test syntactic equalities whereas we want to consider languages of terms modulo an equational theory. The problem is the same with the tree automata of [DCC95]. Our definition based on Horn clauses and our saturation method solve this problem by considering a class of tree automata which combines both equality constraints like TAD and equational theories like TAE. The tree automata defined this way test equality constraints modulo an equational theory and recognize languages of terms modulo the same theory.

Definition 5 *A tree automaton with equational constraints modulo an equational theory (TADE) is the union of an equational theory and of a TAD (finite set of clauses of type (t) or (d)).*

We show in Appendix E that every reduction automaton with equality constraints only is equivalent to a TADE of the same size, as long as its transitions fulfill the restrictions on predicates introduced in the definition of (t) and (d) in order to make emptiness decidable.

Before we show the decidability of GMP for TADE, let us introduce the following measure on terms: $m_5(u) = (mvar(u), |u|)$ where $mvar(u)$ is the multiset of the numbers of occurrences for each variable in u and $|u|$ is the size of u, that is the number of symbols in u. The measure of terms are compared using a lexicographic extension of orderings for each component. We denote by $|u|_z$ the number of occurrence of symbol z in term u.

Lemma 2 *We consider terms s, t and a variable x such that $vars(s) \cap vars(t) = \emptyset$, s is not a variable, t is linear, $x \in vars(t)$, $x \neq t$, and σ is an mgu of s and t. Then $m_5(x\sigma)$ is strictly less than $m_5(s)$.*

This lemma is proved in Appendix F.

Proposition 6 *GMP for TADE modulo a $\succ$-convergent sublinear and collapsing equational theory can be solved using basic ordered paramodulation with selection and ε-splitting.*

Proof. We assume given a TADE $\mathcal{A}$ and a goal clause $Q(t) \Rightarrow$. We note $\mathcal{R}$ the TRS corresponding to the clauses of type (eq) of $\mathcal{A}$. We will show the termination of the saturation of $\mathcal{A} \cup \{Q(t) \Rightarrow\}$ under basic ordered paramodulation wrt the ordering $\succ_{lpo}$ and the selection function sel_2 (defined in the proof of Proposition 4) and with eager ε-splitting.

The measure of a clause $C = \Gamma, \Xi \Rightarrow H[\![\theta]\!]$, where Γ is a multiset of non-equational atoms and Ξ is a multiset of equations, is the tuple $(m_1(C), \ldots, m_6(C))$ where:

- $m_2(C)$, $m_4(C)$, $m_6(C)$ are defined like in the proof of Proposition 4 (applying the definitions to the expansion of clauses),
- $m_1(C) = \infty$ if H is an equation, and is defined like in the proof of Proposition 4 otherwise,
- $m_3(C)$ is the number of function symbols in Γ, Ξ and H. Note that we consider here the number of function symbols without applying the constraint θ,
- $m_5(C) = m_5(s)$ if $H = Q(s)$, and $m_5(C) = (\{\}, 0)$ is the other cases.

The strict ordering $\gg$ on measures is defined as the lexicographic extension: $\left(>, >, >, \succ^{mul}, (>^{mul}, >), \succ_{lpo}^{mul}\right)^{lex}$, where $>$ denotes the ordering on $\mathbb{N}$.

The following Fact permits to conclude the proof of the proposition. It refers to the following clause type (df) similar to (gf):

$$Q_1(y_{i_1}), \ldots, Q_k(y_{i_k}), \underline{Q'_1(f(y_1, \ldots, y_n))}, \ldots, \underline{Q'_m(f(y_1, \ldots, y_n))} \Rightarrow Q(r) \qquad \text{(df)}$$

where $m \geq 0$, $n > 0$, $y_1, \ldots, y_n$ are distinct variables, $i_1, \ldots, i_k \leq n$, r is either one of the y_i, with $i \leq n$, or $f(y_1, \ldots, y_n)$, and if Q is a test predicate then every test predicate among $Q_1, \ldots, Q_k, Q'_1, \ldots, Q'_m$ is strictly smaller than Q (wrt $\succ$), otherwise, $Q_1, \ldots, Q_k, Q'_1, \ldots, Q'_m \in \mathcal{P}_0$.

inf.	1^{st} pr.	2^{nd} pr.	cl	1^{st} premise						2^{nd} premise					
				m_1	m_2	m_3	m_4	m_5	m_6	m_1	m_2	m_3	m_4	m_5	m_6
Eq	d/d'	$/$	d'/d_+	$=$	$>$	_	_	_	_						
RP	eq	t	l_+	$>$	_	_	_	_	_	$=$	$=$	$>$	_	_	_
LP	eq	d/d'	d'	$>$	_	_	_	_	_	$=$	$=$	$>$	_	_	_
LP	eq	g_+	g_+	$>$	_	_	_	_	_	$=$	$=$	$>$	_	_	_
R	_	t	$/$												
R	t	$\mathsf{l}_+{}^{(1)}$	l_+	$=$	$=$	$>$	_	_	_	$=$	$=$	$=$	$=$	$=$	$>$
R	t	$\mathsf{d}_+{}^{(1)}$	d_+	$>$	_	_	_	_	_	$=$	$=$	$=$	$\geq$	$=$	$>$
R	t	$\mathsf{d}_+{}^{(2)}$	df												
R	t	df	df												
R	t	$\mathsf{g}_+{}^{(1)}$	g_+	$>$	_	_	_	_	_	$=$	$=$	$\geq$	$=$	$=$	$>$
R	t	$\mathsf{g}_+{}^{(2)}$	gf												
R	t	gf	gf												
R	l_+	_	$/$												
R	d_+/df	l_+	d_+	$=$	$=$	$=$	$=$	$>$	_	$=$	$=$	$=$	$>$	_	_
R	d_+/df	d_+/df	d_+	$>$	_	_	_	_	_	$=$	$=$	$=$	$>$	_	_
R	d_+/df	g_+/gf	g_+	$>$	_	_	_	_	_	$=$	$=$	$\geq$	$>$	_	_
R	sp	$\mathsf{l}_+{}^{(3)}$	l_+	$>$	_	_	_	_	_	$=$	$=$	$=$	$>$	_	_
R	sp	$\mathsf{d}_+{}^{(3)}$	d_+	$>$	_	_	_	_	_	$=$	$=$	$=$	$>$	_	_
R	sp	$\mathsf{g}_+{}^{(3)}$	g_+	$>$	_	_	_	_	_	$=$	$=$	$=$	$>$	_	_
R	sp	$\mathsf{g}_+{}^{(3)}$	sp												
εsplit	l_+		gf												
εsplit	l_+		l_+	$=$	$=$	$=$	$\geq$	$=$	$>$						
εsplit	d_+		gf												
εsplit	d_+		d_+	$=$	$=$	$=$	$\geq$	$=$	$>$						
εsplit	g_+		gf												
εsplit	g_+		g_+	$=$	$=$	$=$	$\geq$	$=$	$>$						

(1) no neg. splitting literals and at least one literal selected
(2) no literal selected (3) at least one negative splitting literal (selected)

Figure 2: Case analysis in the proof of Proposition 6.

Fact 2 Starting from $\mathcal{A}$ and a goal $Q(t) \Rightarrow$, every step of basic ordered paramodulation and ε-splitting returns either a clause smaller than all its premises (wrt $\gg$) or a clause of type (df) or (gf).

Fact 2 is proved by a case analysis summarized in Figure 2 and detailed in Appendix G. Let us just define here a new type of clauses appearing during the saturation. RP with an equational clause $\Rightarrow f(\ell_1, \ldots, \ell_n) \to r$ of type (eq) into a clause of type (t) returns a clause expanded into the following form[4]:

$$\underline{q_1}, \ldots, \underline{q_k}, Q_1(\ell_1), \ldots, Q_n(\ell_n) \Rightarrow Q(r) \qquad (\mathsf{l}_+)$$

where $n \geq 0$, $\ell_1, \ldots, \ell_n$ are linear, r is either ground or a variable x, $q_1, \ldots, q_k \in \mathcal{Q}$, and either $Q_1, \ldots, Q_n, Q \in \mathcal{P}_0$ or Q is a test predicate and at most one of $Q_1, \ldots, Q_n$ is equal to Q, and the others belong to $\mathcal{P}_0$. □

8 Verification of Reachability in a Fragment of the Applied pi Calculus

First-order theorem proving techniques have been applied for a long time to the formal verification of security protocols. One famous example is the system ProVerif [Bla04] which has been applied to the verification of various non trivial properties on many real protocols. We believe that the saturation results presented in this

[4]Note that no further applications of RP or LP other than resolution is possible into the clause obtained, because of the basic strategy.

paper can help in this setting, and illustrate this claim with the verification of a reachability property for a small protocol, wrt a bounded number of sessions.

We describe here a simplification (without certificates and timestamps) of a key distribution protocol of Denning & Sacco [DS81] for a symmetric key exchange in an asymmetric cryptosystem. We shall use for its specification a fragment of the applied pi-calculus [AF01], a generalization of pi-calculus where messages are terms modulo an equational theory, presented by a rewrite system. We shall consider in our example sublinear–collapsing rewrite rules for projections on pairs: $\mathsf{fst}(\mathsf{pair}(x,y)) \to x$ and $\mathsf{snd}(\mathsf{pair}(x,y)) \to y$, decryption in symmetric key cryptography: $\mathsf{dec}\big(\mathsf{enc}(x,y),y\big) \to x$ (the variables x and y correspond respectively to the encrypted plaintext and the encryption key), decryption in public (asymmetric) key cryptography: $\mathsf{adec}\big(\mathsf{aenc}(x,y),\mathsf{inv}(y)\big) \to x$ and $\mathsf{adec}\big(\mathsf{aenc}(x,\mathsf{inv}(y)),y\big) \to x$ where inv is an idempotent operator, following the rule $\mathsf{inv}(\mathsf{inv}(y)) \to y$, which associates to a public encryption key its corresponding private key (for decryption), and conversely. The symbol inv is called *secret* and all the others are called *public*. The protocol is described using processes in the following syntax:

P	:=	$null$	null process
		$q(x).P$	message input on channel q (x is a variable *bounded* in P)
		$\overline{q}\langle t\rangle.P$	message output on channel q
		$if\ u_1 = v_1, \ldots, u_k = v_k\ then\ P$	conditional (u_i, v_i are terms)
		$p.P$	program point (p is a natural number)
		$P \mid P'$	parallel composition

The program points have no special meaning, they are just added for technical convenience. We do not allow replication of processes $P!$ since we focus here on finite processes. Note however that our formalism would also permit us to deal with an unbounded number of sessions (hence $P!$), considering upper-approximations of reachable state sets like *e.g.* [GK00, NRNS02, Mon03, Bla04].

For the Denning & Sacco protocol, we assume one public channel q and an initial configuration made of two concurrent agents $agent_A \mid agent_B$ with:

$$agent_A \equiv 0.\overline{q}\big\langle \mathsf{pair}(A, \mathsf{aenc}(\mathsf{aenc}(K, \mathsf{inv}(\mathsf{pub}(A))), \mathsf{pub}(B)))\big\rangle.1$$

$$agent_B \equiv 0.q(x).\overline{q}\big\langle \mathsf{enc}(S, \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(x))))\big\rangle.1$$

The agent A sends to agent B a freshly chosen symmetric key K for further secure communications (A, B, K, S are constant function symbols). This key is encrypted, for authentication purpose, using the asymmetric encryption function aenc and the secret key of A, represented as the inverse $\mathsf{inv}(\mathsf{pub}(A))$ of its public key $\mathsf{pub}(A)$. The result of this encryption is later encrypted with B's public key $\mathsf{pub}(B)$ so that only B shall be able to learn K. Moreover, A appends its name at the beginning of the message (using the pairing function pair) so that the receiver B knows which public key to use in order to obtain K.

The agent B, while reading a message x, expects that x has the above form. Then, he extracts the symmetric key K, applying twice the asymmetric decryption function adec to the second component of x, obtained by application of the projection function snd. This key K is then used by agent B to encrypt (with the function enc) a secret code S that he wants to communicate to the agent A. This behaviour of the agents can be described by transitions of the labeled operational semantics of [AF01] for processes that interact with an insecure environment (called *the attacker* below). Sending message s to the network corresponds to the transition: $\overline{q}\langle s\rangle.P \xrightarrow{\nu y.\overline{q}\langle y\rangle} \{y = s\} \mid P$, where y is a fresh variable, and $\{y = s\}$ is a *floating substitution*, an addition to the above syntax used to keep track of the messages read by the attacker. Reading the message t corresponds to $q(x).P \xrightarrow{q(t)} P\{x \mapsto t\}$. If q is a public channel (in which the attacker may write), t is a message built by the attacker, *i.e.* it is made of the variables y collected as above and public function symbols. Otherwise, t is a variable y (a message is transmitted without tampering).

We consider states of the system made of both agents where each of them is either at program point 0 or 1. For the modeling of this system with a TADE $\mathcal{A}$, we use four predicates $Q_{00} \in \mathcal{P}_0$ and Q_{01}, Q_{10}, Q_{11}, test predicates, where Q refers to the contents of channel q and the pairs of program points correspond to the respective states of $agent_A$ and $agent_B$.

The process $agent_A$ is modeled by the following clauses of $\mathcal{A}$ of type (d), where i is either 0 or 1:

$$Q_{0i}(x) \Rightarrow Q_{1i}(\mathsf{pair}(A, \mathsf{aenc}(\mathsf{aenc}(K, \mathsf{inv}(\mathsf{pub}(A))), \mathsf{pub}(B))))$$

In this transition the state of $agent_A$ is upgraded from 0 to 1 while the state of $agent_B$ is left unchanged. The process $agent_B$ is modeled by the following clauses of $\mathcal{A}$ of type (d), with $i = 0, 1$:

$$Q_{i0}(x) \Rightarrow Q_{i1}(\mathsf{enc}(S, \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(x)))))$$

There exist some protocols for which the transitions cannot be modeled this way. Indeed, the head of the clause (message sent) may contain the message x which has just been received, like for $agent_B$, but no other message received previously. In order to deal with this situation, we need to extend the class TADE with non rewritable memories, which are additional arguments of the predicates and can be updated only once by (d)-like transitions. The saturation still terminates for this class. The formal study of this class is in preparation for the long version of this paper.

We now add some clauses of type (t) and (d) to $\mathcal{A}$ in order to model the control of an attacker over the communication channel q. Since this channel is public, we assume that the attacker is able to read and write any message on it. He is also able to forge new messages by application of public functions to the messages collected. Hence, for each public function symbol, for instance pair, we have the following clauses:

$$\begin{array}{rcl@{\qquad}rcl}
Q_{00}(x_1), Q_{00}(x_2) & \Rightarrow & Q_{00}(\mathsf{pair}(x_1,x_2)) & Q_{00}(x_1), Q_{01}(x_2) & \Rightarrow & Q_{01}(\mathsf{pair}(x_1,x_2)) \\
Q_{00}(x_1), Q_{10}(x_2) & \Rightarrow & Q_{10}(\mathsf{pair}(x_1,x_2)) & Q_{00}(x_1), Q_{11}(x_2) & \Rightarrow & Q_{11}(\mathsf{pair}(x_1,x_2)) \\
Q_{01}(x_1), Q_{00}(x_2) & \Rightarrow & Q_{01}(\mathsf{pair}(x_1,x_2)) & Q_{01}(x_1), Q_{10}(x_2), x = \mathsf{pair}(x_1,x_2) & \Rightarrow & Q_{11}(x) \\
Q_{10}(x_1), Q_{00}(x_2) & \Rightarrow & Q_{10}(\mathsf{pair}(x_1,x_2)) & Q_{10}(x_1), Q_{01}(x_2), x = \mathsf{pair}(x_1,x_2) & \Rightarrow & Q_{11}(x)
\end{array}$$

Note that in the above clauses we allow several combinations of agent's states in the antecedents, but not every combination. The principle is that if $agent_A$ (resp. $agent_B$) is in state 1 in the first antecedent, it must be in state 0 in the second one (and conversely), because we assume that each agent can run only once. We can also add clauses ensuring that some ground terms initially known to the attacker belong to $L(\mathcal{A}, Q_{00})$, like: $\Rightarrow Q_{00}(A)$. Finally, $\mathcal{A}$ contains of course the six rewrite rules presented above, specified as equational clauses of type (eq).

The above version of the Denning-Sacco protocol is flawed: there exists an attack which involves $agent_B$ only and reveals the secret S. Assume that the attacker initially knows A, $\mathsf{pub}(A)$, $\mathsf{pub}(B)$. This situation can be modeled as above by a regular subset of $L(\mathcal{A}, Q_{00})$ in our framework. The attacker is able to construct and send the message $m_0 = \mathsf{pair}(A, \mathsf{aenc}(A, \mathsf{pub}(B)))$ on the channel q, because this term is made of terms of its initial knowledge and public function symbols pair and aenc. Indeed, we have $m_0 \in L(\mathcal{A}, Q_{00})$. When the agent B reads m, he replies (in channel q) with $\mathsf{enc}(S, \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(m), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(m))))$. Using the above rewrite rules for projection, this term is reduced to $\mathsf{enc}(S, \mathsf{adec}(\underline{\mathsf{adec}(\mathsf{aenc}(A, \mathsf{pub}(B)), \mathsf{inv}(\mathsf{pub}(B)))}, \mathsf{pub}(A)))$ and with one of the above rewrite rules for asymmetric decryption applied to the underline redex, this term is further reduced to $m_1 = \mathsf{enc}(S, \mathsf{adec}(A, \mathsf{pub}(A)))$. It means that $m_1 \in L(\mathcal{A}, Q_{01})$. The attacker is then able to construct the "key" $\mathsf{adec}(A, \mathsf{pub}(A))$, which belongs to $L(\mathcal{A}, Q_{00})$, and can recover S by decryption. Indeed, thanks to the clause $Q_{01}(x_1), Q_{00}(x_2) \Rightarrow Q_{01}(\mathsf{dec}(x_1, x_2))$, we have $S \in L(\mathcal{A}, Q_{01})$. It means that the secret S is revealed on channel q.

The protocol can be patched in order to avoid such an attack, by requiring the agent B to make some preliminary verifications on the message received before sending his answer:

$$\begin{array}{lll@{\quad}rcl}
agent'_B \equiv 0.q(x). & \mathit{if} & \mathsf{snd}(\mathsf{fst}(\mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(x))))) & = & x_B, \\
 & & \mathsf{fst}(\mathsf{fst}(\mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(x))))) & = & \mathsf{fst}(x) \\
 & \mathit{then} & \overline{q}\langle \mathsf{enc}(S, \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(x))))\rangle.1 & &
\end{array}$$

This feature can be modeled in TAD by the following clause of type (d):

$$\begin{array}{rcl@{\quad}l}
Q_{i0}(x), \quad x_B & = & \mathsf{snd}(\mathsf{fst}(\mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(x))))), & \\
\mathsf{fst}(x) & = & \mathsf{fst}(\mathsf{fst}(\mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(x)))))) & \Rightarrow Q_{i1}(m[x])
\end{array}$$

where $m[x] = \mathsf{enc}(S, \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(x))))$ and i is either 0 or 1.

9 Conclusion and further works

We have introduced new classes of tree automata with constraints and shown that the General Membership Problem is decidable for them with a uniform theorem-proving technique. Potential extensions are numerous.

As future work we think to extend the tree automata classes defined in this paper to disequality tests as in [DCC95]. This would permit us to characterize languages of normal form wrt a TRS and is useful in particular in inductive theorem proving [CDG+97]. Such an extension is not trivial. In particular, it is not possible to simply add the disequations as positive (equational) literals to the clauses defined above. This would lead us out of the expected semantics. Some ad-hoc axiomatization of the disequality, according to the equational theory, is necessary. Other tests, for instance ordering tests, could also be investigated.

Equality tests between brother positions à la [BT92] can be easily incorporated into the Horn clauses representation of tree automata (see *e.g.* [JMW98]). Equations are not necessary for this purpose, since multiple occurrences of a variable suffice, as in: $Q_1(x), Q_2(x) \Rightarrow Q\big(f(x,x)\big)$ The combination of TA classes of [BT92] and [DCC95] preserves emptiness decidability [CCC+94]. Hence the combination of the above class of TA with equality test modulo and unrestricted test between brother positions is interesting to study.

It would be interesting too to extend the above saturation results (in particular for classes modulo monadic or collapsing theories) to term algebra modulo AC, using AC-paramodulation techniques. This combination (AC + sublinear–collapsing) permits to axiomatize primitives like the XOR.

Finally it should be possible to extend our approach to the recognition of binary relations on terms by automata. This would help to characterize some equivalences in the pi calculus and to capture for instance guessing attacks expressed using static equivalence. However, one should be very careful with this approach since combining binary relations and equality tests leads quickly to undecidability [Tre00].

Acknowledgments. We wish to thank Stéphanie Delaune for her contribution to early phases of this work and Jean Goubault-Larrecq and Christopher Lynch for their help about ε-splitting.

References

[AF01] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 104–115, 2001.

[AL00] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Concurrency Theory, 11th Int. Conf. on Concurrency Theory, CONCUR*, volume 1877 of *LNCS*, pages 380–394. Springer, 2000.

[BGLS95] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic Paramodulation. *Information and Computation*, 121(2):172–192, 1995.

[Bla04] B. Blanchet. Automatic Proof of Strong Secrecy for Security Protocols. In *IEEE Symp. on Security and Privacy*, pages 86–100, 2004.

[BT92] B. Bogaert and S. Tison. Equality and Disequality Constraints on Direct Subterms in Tree Automata. In *9th Symp. on Theoretical Aspects of Computer Science, STACS*, volume 577 of *LNCS*, pages 161–171. Springer, 1992.

[CCC+94] A.-C. Caron, H. Comon, J.-L. Coquidé, M. Dauchet, and F. Jacquemard. Pumping, Cleaning and Symbolic Constraints Solving. In *21st Int. Coll. on Automata, Languages and Programming, ICALP*, volume 820 of *LNCS*, pages 436–449. Springer, 1994.

[CDG+97] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications.* `http://www.grappa.univ-lille3.fr/tata`, 1997.

[DCC95] M. Dauchet, A.-C. Caron, and J.-L. Coquidé. Automata for Reduction Properties Solving. *Journal of Symbolic Computation*, 20(2):215–233, 1995.

[DJ90] N. Dershowitz and J.-P. Jouannaud. *Rewrite systems*, chapter Handbook of Theoretical Computer Science, Volume B, pages 243–320. Elsevier, 1990.

[DS81] D. E. Denning and G. M. Sacco. Timestamps in Key Distribution Protocols. In *Communications of the ACM*, 1981.

[FSVY91] T. Frühwirth, E. Shapiro, M. Vardi, and E. Yardeni. Logic programs as types for logic programs. In *Proc. of the 6th IEEE Symposium on Logic in Computer Science*, pages 300–309, 1991.

[GK00] Th. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Proc. of 17th Int. Conf. on Automated Deduction, CADE*, volume 1831 of *LNCS*. Springer, 2000.

[Gou05] J. Goubault-Larrecq. Deciding $\mathcal{H}_1$ by Resolution. *Information Processing Letters*, 95(3):401–408, 2005.

[JMW98] F. Jacquemard, Ch. Meyer, and Ch. Weidenbach. Unification in Extensions of Shallow Equational Theories. In *9th Int. Conf. on Rewriting Techniques and Applications, RTA*, volume 1379 of *LNCS*, pages 76–90. Springer, 1998.

[KW04] R. Küsters and T. Wilke. Automata-Based Analysis of Recursive Cryptographic Protocols. In *21st Annual Symp. on Theoretical Aspects of Computer Science, STACS*, volume 2996 of *LNCS*, pages 382–393. Springer, 2004.

[LS] S. Limet and G. Salzer. Basic Rewriting via Logic Programming with an Application to the Reachability Problem. *Journal of Automata, Languages and Combinatorics*. To appear.

[LS02] D. Lugiez and Ph. Schnoebelen. The Regular Viewpoint on PA-Processes. *Theoretical Computer Science*, 274(1-2):89–115, 2002.

[Mon81] J. Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG*. PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, UST, Villeneuve d'Ascq, France, 1981.

[Mon03] D. Monniaux. Abstracting cryptographic protocols with tree automata. *Science of Computer Programming*, 47(2–3):177–202, 2003.

[NR01] Robert Nieuwenhuis and Albert Rubio. *Paramodulation-Based Theorem Proving*, chapter Handbook of Automated Reasoning, Volume I, Chapter 7. Elsevier Science and MIT Press, 2001.

[NRNS02] F. Nielson, H. Riis Nielson, and H. Seidl. Normalizable Horn Clauses, Strongly Recognizable Relations, and Spi. In *Static Analysis, 9th Int. Symp., SAS*, volume 2477 of *LNCS*, pages 20–35. Springer, 2002.

[OT02] H. Ohsaki and T. Takai. Decidability and Closure Properties of Equational Tree Languages. In *13th Int. Conf. on Rewriting Techniques and Applications, RTA*, volume 2378 of *LNCS*, pages 114–128. Springer, 2002.

[RV01] A. Riazanov and A. Voronkov. Splitting Without Backtracking. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence, IJCAI*, pages 611–617. Morgan Kaufmann, 2001.

[STFK02] H. Seki, T. Takai, Y. Fujinaka, and Y. Kaji. Layered Transducing Term Rewriting System and Its Recognizability Preserving Property. In *Int. Conf. on Rewriting Techniques and Applications, RTA*, volume 2378 of *LNCS*, pages 98–113. Springer, 2002.

[Tis00] S. Tison. Tree automata and term rewrite systems. Invited tutorial at the 11th Int. Conf. on Rewriting Techniques and Applications, RTA, 2000.

[Tre00] R. Treinen. Predicate logic and tree automata with tests. In J. Tiuryn, editor, *Proc. of the 3rd Int. Conf. on Foundations of Software Science and Computation Structures, FoSSaCS*, volume 1784 of *LNCS*, pages 329–343. Springer, 2000.

[Ver03] K. N. Verma. *Two-Way Equational Tree Automata*. PhD thesis, ENS Cachan, September 2003.

Appendix A Proof of Proposition 1

The different cases of resolution steps between clauses of type (s), (gs) and (gf) are listed below:

R(_, s): no resolution step is possible into a clause of type (s) because of the maximality condition (v) in LP. Indeed, no literal is selected by sel_1 in any clause of the form $Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big)$ and, for all $i \leq n$, we have $Q\big(f(x_1, \ldots, x_n)\big) \succ_{lpo} Q_i(x_i)$. Hence, for all substitution σ, $Q_i(x_i\sigma)$ cannot be maximal among $Q_1(x_1\sigma), \ldots, Q_n(x_n\sigma)$, $Q\big(f(x_1, \ldots, x_n)\sigma\big)$.

R(s, gs) returns a clause of type (gs) when one non-splitting negative literal $P_1(s_1)$ in the premise (gs) is selected by sel_1, *i.e.* when $s_1 = f(s'_1, \ldots, s'_n)$ (note that in this case the premise does not contain splitting literals by definition of sel_1):

$$\frac{Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow P_1\big(f(x_1, \ldots, x_n)\big) \quad \underline{P_1(s_1)}, \ldots, P_m(s_m) \Rightarrow [\,q\,]}{Q_1(s'_1), \ldots, Q_n(s'_n), P_2(s_2), \ldots, P_m(s_m) \Rightarrow [\,q\,]}\,\mathsf{R}$$

R(s, gs) returns a clause of type (gf) when no negative literal is selected by sel_1 in the premise (gs). Note that in this case, this premise contains only one variable, otherwise it would be split. The tuple $x_1, \ldots, x_n$ is denoted $\overline{x}$ below:

$$\frac{Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow P_1\big(f(x_1, \ldots, x_n)\big) \quad P_1(y), \ldots, P_k(y) \Rightarrow [\,q\,]}{Q_1(x_1), \ldots, Q_n(x_n), \underline{P_2(f(\overline{x}))}, \ldots, \underline{P_k(f(\overline{x}))} \Rightarrow [\,q\,]}\,\mathsf{R}$$

R(s, gf) returns a clause of type (gf) when one non-splitting negative literal at least is selected by sel_1 in the premise (gf) – the tuple $y_1, \ldots, y_n$ is denoted $\overline{y}$:

$$\frac{\begin{array}{c} Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow P'_1\big(f(x_1, \ldots, x_n)\big) \\ P_1(y_{i_1}), \ldots, P_k(y_{i_k}), \underline{P'_1(f(\overline{y}))}, \ldots, \underline{P'_m(f(\overline{y}))} \Rightarrow [\,q\,] \end{array}}{P_1(y_{i_1}), \ldots, P_k(y_{i_k}), Q_1(y_1), \ldots, Q_n(y_n), \underline{P'_2(f(\overline{y}))}, \ldots, \underline{P'_m(f(\overline{y}))} \Rightarrow [\,q\,]}\,\mathsf{R}$$

R(s, gf) returns a clause of type (gf) when no negative literal is selected by sel_1 in the premise (gf). This case in embedded R(s, gs) when the second premise has type (gs) and no selected literals, which has been treated above.

R(sp, gs) returns a clause of type (gs).

$$\frac{\Rightarrow q_1 \quad \underline{q_1}, \ldots, \underline{q_k}, P_1(s_1), \ldots, P_n(m_m) \Rightarrow [\,q\,]}{\underline{q_2}, \ldots, \underline{q_k}, P_1(s_1), \ldots, P_m(s_m) \Rightarrow [\,q\,]}\,\mathsf{R}$$

Note that this is the only case where a clause of type (gs) or (gf) can be involved as first premise in a resolution step.

Appendix B Proof of Proposition 3

As in the proof of Proposition 2, we reduce the halting problem of a 2-counter machine $\mathcal{M}$. We consider the same representation of the configurations of $\mathcal{M}$ as in Proposition 2, using in particular the same signature. The respective initial and final states of $\mathcal{M}$ are q_0 and q_f. We construct below a non-deterministic reduction automaton $\mathcal{A}$ which will accepts in the language $L(\mathcal{A}, Q_\mathsf{f})$ the term representations of halting computations of $\mathcal{M}$, starting with the configuration $q_0(0, 0)$ and ending with $q_\mathsf{f}(i_1, i_2)$ for some i_1 and i_2.

We have in $\mathcal{A}$ two states Q_0 (for 0) and Q_1 (for strictly positive integers), with the transitions: $\Rightarrow Q_0(0)$, $Q_0(x) \Rightarrow Q_1(s(x))$, $Q_1(x) \Rightarrow Q_1(s(x))$. Below, Q_N is an abbreviation for either Q_0 or Q_1.

The transition $T_1 = c_1 \rightarrow d_1$ of $\mathcal{M}$, with $c_1 = q(x, y)$, $d_1 = q'(s(x), y)$ (it corresponds to the machine instruction q: ADD 1 TO COUNTER 1; GOTO q') is represented by the term $g(q(x, y), h(g(q'(s(x), y), u), u))$ where u is the rest of the computation (the term is displayed in Figure 3 for sake of readability). We use the following states and clauses for the recognition of this term T_1:

- $Q_N(x_1), Q_N(x_2) \Rightarrow Q_{c_1}(q(x_1, x_2))$ and $Q_1(x_1), Q_N(x_2) \Rightarrow Q_{d_1}(q'(x_1, x_2))$ where the states Q_{c_1} Q_{d_1} are respectively associated to the left and right members of the transition,

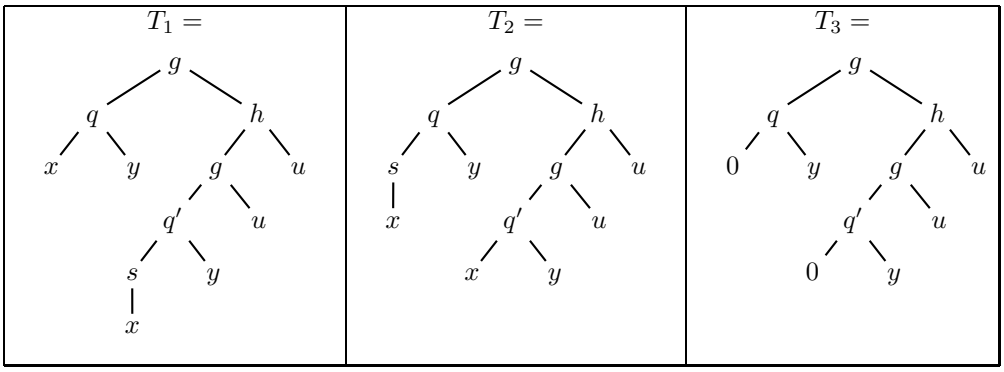

Figure 3: Transitions of the 2-counter machine

- $Q_{d_1}(x_1), Q_\forall(x_2) \Rightarrow Q_{gd_1}(g(x_1, x_2))$,
- $Q_\forall$ is a "universal" state, $\Rightarrow Q_\forall(0)$, $Q_\forall(x) \Rightarrow Q_\forall(s(x))$, $Q_\forall(x_1), Q_\forall(x_2) \Rightarrow Q_\forall(f(x_1, x_2))$ for every binary function symbol f among g, h, k or any state q of $\mathcal{M}$.
- $Q_{gd_1}(x_1), Q_\forall(x_2) \Rightarrow Q_{hd_1}(h(x_1, x_2))$,
- and this last clause which permits to accept the term T_1 in Figure 3 into a state also called T_1 (*i.e.* in $L(\mathcal{A}, T_1)$), and performs equality tests:

$$Q_{c_1}(x_1), Q_{hd_1}(x_2) \Rightarrow T_1(g(x_1, x_2))[\![x_1|_1 = x_2|_{1111}, x_1|_2 = x_2|_{112}, x_2|_2 = x_2|_{12}]\!]$$

The transition $T_2 = c_2 \rightarrow d_2$ of $\mathcal{M}$, with $c_2 = q(s(x), y)$, $d_2 = q'(x, y)$ (it corresponds to the machine instruction q: IF COUNTER 1 $\neq 0$ DEC 1; GOTO q') is represented by the term $g(q(s(x), y), h(g(q'(x, y), u), u))$ (see Figure 3). We use the following states and clauses for the recognition of this term T_2:

- $Q_1(x_1), Q_N(x_2) \Rightarrow Q_{c_2}(q(x_1, x_2))$ and $Q_N(x_1), Q_N(x_2) \Rightarrow Q_{d_2}(q'(x_1, x_2))$,
- $Q_{d_2}(x_1), Q_\forall(x_2) \Rightarrow Q_{gd_2}(g(x_1, x_2))$, $Q_{gd_2}(x_1), Q_\forall(x_2) \Rightarrow Q_{hd_2}(h(x_1, x_2))$,
- and the last clause (for the recognition of the term T_2, Figure 3) which performs equality tests:

$$Q_{c_2}(x_1), Q_{hd_2}(x_2) \Rightarrow T_2(g(x_1, x_2))[\![x_1|_{11} = x_2|_{111}, x_1|_2 = x_2|_{112}, x_2|_2 = x_2|_{12}]\!]$$

The transition $T_3 = c_3 \rightarrow d_3$ of $\mathcal{M}$, with $c_3 = q(0, y)$, $d_3 = q'(0, y)$ (it corresponds to the machine instruction q: IF COUNTER 1 $= 0$; GOTO q') is represented by the term $g(q(0, y), h(g(q'(0, y), u), u))$ (see Figure 3). We use the following states and clauses for the recognition of this term T_3:

- $Q_0(x_1), Q_N(x_2) \Rightarrow Q_{c_3}(q(x_1, x_2))$ and $Q_0(x_1), Q_N(x_2) \Rightarrow Q_{d_3}(q'(x_1, x_2))$,
- $Q_{d_3}(x_1), Q_\forall(x_2) \Rightarrow Q_{gd_3}(g(x_1, x_2))$, $Q_{gd_3}(x_1), Q_\forall(x_2) \Rightarrow Q_{hd_3}(h(x_1, x_2))$,
- and the last clause (for the recognition of the term T_3, Figure 3) which performs equality tests:

$$Q_{c_3}(x_1), Q_{hd_3}(x_2) \Rightarrow T_3(g(x_1, x_2))[\![x_1|_2 = x_2|_{112}, x_2|_2 = x_2|_{12}]\!]$$

To model the chaining of transitions, we use a a new state Q and, for each transition T of $\mathcal{M}$ and associated state T we introduce the following unconstrained clauses (recall that T is the state symbol associated to the transition as above):

$$T(x_1), Q(x_2) \Rightarrow Q(h(x_1, x_2))$$

We have a special constrained clause, associated to the unique transition T_0 of $\mathcal{M}$ starting from the initial configuration $c_0 = q_0(0, 0)$, ($\mathcal{M}$ is assumed deterministic). Note that in this clause we have a symbol k in the head, instead of a h:

$$T_0(x_1), Q(x_2) \Rightarrow Q_{\mathsf{f}}(k(x_1, x_2))[\![x_1|_2 = x_2]\!]$$

Finally, we consider three unconstrained clauses to initiate the bottom-up computation of the automaton with a final configuration $q_{\mathsf{f}}(i_1, i_2)$ of $\mathcal{M}$. We assume wlog that the state q_{f} can not be reentered by $\mathcal{M}$. These clauses aim at accepting the term $h(g(q_{\mathsf{f}}(i_1, i_2), c), c)$ in the language $L(\mathcal{A}, Q)$:

$$\begin{array}{rcl@{\qquad}rcl} & \Rightarrow & Q_c(c), & Q_N(x_1), Q_N(x_2) & \Rightarrow & Q_{c_{\mathsf{f}}}(q_{\mathsf{f}}(x_1, x_2)), \\ Q_{c_{\mathsf{f}}}(x_1), Q_c(x_2) & \Rightarrow & Q_{gc_{\mathsf{f}}}(g(x_1, x_2)), & Q_{gc_{\mathsf{f}}}(x_1), Q_c(x_2) & \Rightarrow & Q(h(x_1, x_2)) \end{array}$$

A example of computation of $\mathcal{A}$ is described in Figure 4. In this figure, the nodes of a recognized term are decorated with the states of $\mathcal{A}$ in which they are accepted. Note that equality test are performed by $\mathcal{A}$ only at nodes labelled with the symbol g.

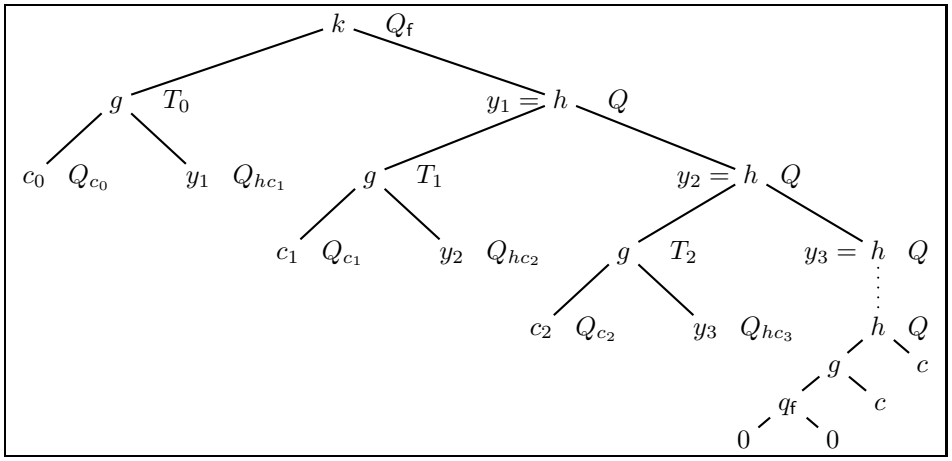

Figure 4: A computation of $\mathcal{A}$

We can show that $\mathcal{M}$ halts on $q_{\mathsf{f}}(i_1, i_2)$ starting from $q_0(0, 0)$ iff $L(\mathcal{A}, Q_{\mathsf{f}}) \neq \emptyset$.
For the only if direction, we associate to a halting computation of $\mathcal{M}$ a tree of $L(\mathcal{A}, Q_{\mathsf{f}})$ as in Figure 4.
For the if direction, we use the following fact:

Fact 3 if i. $t \in L(\mathcal{A}, Q)$ and $t = h(t_1, t_2)$ or $t = k(t_1, t_2)$, ii. $t_1 \in L(\mathcal{A}, T)$ for some transition T, and iii. $t_1|_2 = t_2$, then either the state of the right-hand side of T is q_{f} and $t_2 = h(g(q_{\mathsf{f}}(0, 0), c), c)$, or, i'. $t_2 = h(t'_1, t'_2) \in L(\mathcal{A}, Q)$, ii'. $t_1 \in L(\mathcal{A}, T')$ for a transition T', and iii'. $t'_1|_2 = t'_2$. Moreover, in this last case, the term $t_1|_1$ is rewritten to $t'_1|_1$ by T (seen as a rewrite rule).

Now, observe that by construction of $\mathcal{A}$, if $t \in L(\mathcal{A}, Q_{\mathsf{f}})$ then $t = k(t_1, t_2)$ and $t_1 = g(q_0(0, 0), t_2)$ and $t_2 = L(\mathcal{A}, Q)$. Hence, every term $t \in L(\mathcal{A}, Q_{\mathsf{f}})$ satisfies the hypotheses *i.*, *ii.*, *iii.* of Fact 3, and with this fact, this ensures that t represents a halting computation of $\mathcal{M}$.

Appendix C Proof of Proposition 4 (Fact 1)

In order to show Proposition 4 and Fact 1 we analyze all the instances of Eq, LP (R) and εsplit involving as premises some clauses of type (t), (d), (d_+), (gf), (sp) or (g_+), and we show that the conclusion of each such an instance is either of type (gf) (or (sp)) or either is smaller than all its premises wrt $\gg$.

Eq(d): equality resolution (Eq) is possible with the equations in clauses of type (d) (each of these equations is selected by sel_2) and every such application of (Eq) makes m_2 decrease. Recall that we defined (d_+) at page 9 as the type of clauses obtained from clauses (d) and which contain no more equations.

R(_, t): no resolution step is possible into a clause of type (t) because of the maximality condition (v) in LP. Indeed, for any clause of the form $Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big)$ no negative literal is selected by sel_2 and, for all $i \leq n$, $Q\big(f(x_1, \ldots, x_n)\big) \succ_{lpo} Q_i(x_i)$ by definition of $\succ_{lpo}$ and (t).

R(_, d_+): no resolution step is neither possible into a clause of type (d_+) for the same reason as above.

R(_, d): with the definition of the selection function sel_2, a clause of type (d) and not of type (d_+) has selected equations (which are the only selected literals). Therefore, no inference other than equality resolution (Eq) (in particular no resolution) is possible into such a clause.

$\mathsf{R}(\mathsf{d}, \_)$: because of the definition of sel_2 also, no resolution of a clause containing an equation into another clause is possible.

$\mathsf{R}(\mathsf{t}, \mathsf{g}_+)$ returns a clause of type (sp) or a clause of type (g_+) smaller than both premises when the premise (g_+) has no negative splitting literal and one negative literal $P_1(s_1)$ selected by sel_2, *i.e.* when $s_1 = f(s'_1, \ldots, s'_n)$. If $n > 0$ or the premise (g_+) has strictly more than one antecedent we obtain a clause (g_+) (the tuple $x_1, \ldots, x_n$ is denoted $\overline{x}$ below):

$$\frac{Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow P_1\big(f(\overline{x})\big) \quad \underline{P_1(s_1)}, \ldots, P_m(s_m) \Rightarrow [\,q\,]}{Q_1(s'_1), \ldots, Q_n(s'_n), P_2(s_2), \ldots, P_m(s_m) \Rightarrow [\,q\,]}\,\mathsf{R}$$

Note that in this case, by definition of (t), the measure m_4 for the conclusion is at most equal to m_4 for the premise (g_+).

If $n = 0$ (f is a constant function symbol) and $m = 1$, we obtain a clause (sp):

$$\frac{\Rightarrow P_1(f) \quad \underline{P_1(s_1)} \Rightarrow [\,q\,]}{\Rightarrow [\,q\,]}\,\mathsf{R}$$

$\mathsf{R}(\mathsf{t}, \mathsf{g}_+)$ returns a clause of type (gf) when no literal is selected by sel_2 in the premise (g_+) (note that it implies that (g_+) has no negative splitting literal). In this case, because of the eager application of ε-splitting, the premise (g_+) contains only one variable x.

$$\frac{Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow P_1\big(f(\overline{x})\big) \quad P_1(x), \ldots, P_m(x) \Rightarrow [\,q\,]}{Q_1(x_1), \ldots, Q_n(x_n), P_1(f(\overline{x})), \ldots, P_m(f(\overline{x})) \Rightarrow [\,q\,]}\,\mathsf{R}$$

$\mathsf{R}(\mathsf{t}, \mathsf{gf})$ returns a clause of type (gf) when the premise of type (gf) has at least one literal selected (the tuples of variables $x_0, \ldots, x_n$ and $y_1, \ldots, y_n$ are respectively denoted $\overline{x}$ and $\overline{y}$ below):

$$\frac{\begin{array}{c} Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow P'_1\big(f(\overline{x})\big) \\ P_1(y_{i_1}), \ldots, P_k(y_{i_k}), \underline{P'_1\big(f(\overline{y})\big)}, \ldots, \underline{P'_m\big(f(\overline{y})\big)} \Rightarrow [\,q\,] \end{array}}{P_1(y_{i_1}), \ldots, P_k(y_{i_k}), Q_1(y_1), \ldots, Q_n(y_n), \underline{P'_2\big(f(\overline{y})\big)}, \ldots, \underline{P'_m\big(f(\overline{y})\big)} \Rightarrow [\,q\,]}\,\mathsf{R}$$

The cases where no literal is selected by sel_2 in the premise of type (gf) is included in the case $\mathsf{R}(\mathsf{t}, \mathsf{g}_+)$, with (g_+) unselected, treated above.

$\mathsf{R}(\mathsf{d}_+, \mathsf{g}_+)$ returns a clause of type (sp) or a clause of type (g_+) smaller than both premises:

$$\frac{Q_1(s_1), \ldots, Q_n(s_n) \Rightarrow P_1(s) \quad P_1(t_1), \ldots, P_m(t_m) \Rightarrow [\,q\,]}{Q_1(s_1\theta), \ldots, Q_n(s_n\theta), P_2(t_2\theta), \ldots, P_m(t_m\theta) \Rightarrow [\,q\,]}\,\mathsf{R}$$

where $\theta = mgu(s, t_1)$. When $n = 0$ and $m = 1$, we obtain a clause of type (sp) or the empty clause.

$\mathsf{R}(\mathsf{d}_+, \mathsf{gf})$ returns a clause of type (sp) or (g_+) smaller than both premises: this case is included in the above case $\mathsf{R}(\mathsf{d}_+, \mathsf{g}_+)$.

$\mathsf{R}(\mathsf{sp}, \mathsf{d}_+)$ returns a clause of type (d_+) smaller than both premises.

$$\frac{\Rightarrow q_1 \quad \underline{q_1}, \ldots, \underline{q_k}, Q_1(s_1), \ldots, Q_n(s_n) \Rightarrow Q(s)}{\underline{q_2}, \ldots, \underline{q_k}, Q_1(s_1), \ldots, Q_n(s_n) \Rightarrow Q(s)}\,\mathsf{R}$$

$\mathsf{R}(\mathsf{sp}, \mathsf{g}_+)$ returns either a clause of type (sp) or a clause of type (g_+) smaller than both premises.

$$\frac{\Rightarrow q_1 \quad \underline{q_1}, [\,\underline{q_2}, \ldots, \underline{q_k}, P_1(s_1), \ldots, P_m(s_m)\,] \Rightarrow [\,q\,]}{[\,\underline{q_2}, \ldots, \underline{q_k}, P_1(s_1), \ldots, P_m(s_m)\,] \Rightarrow [\,q\,]}\,\mathsf{R}$$

Note that the conclusion, when not of type (sp), is indeed smaller than the premise (sp) because $m_1(\mathsf{sp}) = \infty$.

inf.	1^{st} pr	2^{nd} pr	cl	inf.	1^{st} pr	2^{nd} pr	cl	inf.	1^{st} pr	2^{nd} pr	cl
RP	eq	s	I	R	_	s	⁄	R	s	I$^{(1)}$	I
R	s	I$^{(2)}$	f	R	s	f	f	R	sp	I$^{(3)}$	I

(1) no negative splitting literals and at least one literal selected
(2) no literal selected (3) at least one negative splitting literal (selected)

Figure 5: Case analysis in the proof of Proposition 5

εsplit(t) is not possible by definition.

εsplit(d_+): such a step pf ε-splitting replaces a clause of type (d_+) by a clause of type (gf) and a clause of type (d_+) smaller than the premise (B is an ε-block):

$$\frac{q_1, \ldots, q_k, B, Q_1(s_1), \ldots, Q_n(s_n) \Rightarrow Q(s)}{B \Rightarrow q_B \quad q_1, \ldots, q_k, q_B, Q_1(s_1), \ldots, Q_n(s_n) \Rightarrow Q(s)}\ \varepsilon\mathsf{split}$$

εsplit(g_+): this ε-splitting replaces a clause of type (g_+) by a clause of type (gf) and a clause of type (g_+) smaller than the premise (B is an ε-block):

$$\frac{q_1, \ldots, q_k, B, P_1(s_1), \ldots, P_m(s_m) \Rightarrow [q]}{B \Rightarrow q_B \quad q_1, \ldots, q_k, q_B, P_1(s_1), \ldots, P_m(s_m) \Rightarrow [q]}\ \varepsilon\mathsf{split}$$

Appendix D Proof of Proposition 5

Let $\mathcal{S}$ be the smallest set of goal clauses containing the initial goal $P(t) \Rightarrow$ and closed by application of basic left-paramodulation with an equational clause (eq) of $\mathcal{A}$, *i.e.*:

$$\frac{\Rightarrow \ell = r \quad P(s[\ell']_p) \Rightarrow \ [\![\theta]\!] \in \mathcal{S}}{P(s[x]_p) \Rightarrow \ [\![\theta, \ell' = \ell, x = r]\!] \in \mathcal{S}}\ \mathsf{LP}$$

The set $\mathcal{S}$ is finite (of cardinal linear in the size of t and $\mathcal{A}$) because every application of basic left-paramodulation strictly decreases the number of function symbols in the unconstrained part of the goal clause.

We show below by a case analysis that the clauses obtained by application of ordered basic paramodulation with sel_1, $\succ_{lpo}$ and ε-splitting to $\mathcal{A} \cup \{P(t) \Rightarrow\}$ have one of the following types (I) or (f).

$$\underline{q_1}, \ldots, \underline{q_k}, Q_1(s_1), \ldots, Q_n(s_n) \Rightarrow [H] \qquad (\mathsf{I})$$

where $k, n \geq 0$, for every $i \leq n$, either s_i is a subterm of a left hand side of a rule of $\mathcal{R}$, or $Q_i(s_i) \in \mathcal{S}$, $q_1, \ldots, q_k \in \mathcal{Q}$, $Q_1, \ldots, Q_n \in \mathcal{P}_0$, and H is $Q(r)$ for some $Q \in \mathcal{P}_0$ and r is either a variable $x \in vars(s_1, \ldots, s_n)$ a flat term $g(z_1, \ldots, z_m)$ ($k \geq 0$) whose variables $z_1, \ldots, z_m$ belong to $vars(s_1, \ldots, s_n)$ and are pairwise distinct, or H is a splitting literal or else there is no H. Note that (sp) is a subcase of (I).

The following type (f) generalizes the type (gf) defined in the proof of Proposition 1:

$$Q_1(y_{i_1}), \ldots, Q_k(y_{i_k}), \underline{Q'_1\big(f(y_1, \ldots, y_n)\big)}, \ldots, \underline{Q'_m\big(f(y_1, \ldots, y_n)\big)} \Rightarrow [H] \qquad (\mathsf{f})$$

where $k, m \geq 0$, $i_1, \ldots, i_k \leq n$, $y_1, \ldots, y_n$ are distinct variables, $P_1, \ldots, P_k, P'_1, \ldots . P'_m \in \mathcal{P}$, and H is of the form $Q(y_i)$ or $Q\big(f(y_1, \ldots, y_n)\big)$ with $Q \in \mathcal{P}$, or H is a splitting literal or else there is no H, *i.e.* the clause is a goal.

The initial goal $P(t) \Rightarrow$ belongs to type (I) and also belongs to (f) if t is a variable.

The different cases of saturation of $\mathcal{A} \cup \{P(t) \Rightarrow\}$ under basic ordered paramodulation are summarized in the following table and detailed below.

RP(eq, eq) returns a clause of type (eq) which is deleted after simplification by rewriting by $\mathcal{R}$, because by hypothesis, the equational theory of $\mathcal{A}$ is $\succ$-convergent.

RP(eq, s) returns a clause which is expanded into a clause of type (l).

$$\frac{\Rightarrow f(\ell_1,\ldots,\ell_n) = r \quad Q_1(x_1),\ldots,Q_1(x_n) \Rightarrow Q\big(f(x_1,\ldots,x_n)\big)}{Q_1(x_1),\ldots,Q_n(x_n) \Rightarrow Q(y)\ [\![x_1=\ell_1,\ldots,x_n=\ell_n, y=r]\!]}\ \mathsf{RP}$$

Note that no further basic paramodulation step other than resolution is possible into the clauses obtained, because there are no function symbols in its unconstrained part.

LP(eq, s) is not possible because the antecedents of clauses of type (s) contain only variables.

R(_, s): no resolution step is possible into a clause of type (s) because of the maximality condition (v) in LP (see the proof of Proposition 1).

R(s, l) return a clause of type (l) when one non-splitting negative literal $P_1(s_1)$ is selected in the premise of type (l), *i.e.* when $s_1 = f(s'_1,\ldots,s'_n)$ (the tuple $x_1,\ldots,x_n$ is denoted $\overline{x}$ below):

$$\frac{Q_1(x_1),\ldots,Q_n(x_n) \Rightarrow P_1\big(f(\overline{x})\big) \quad \underline{P_1(s_1)},\ldots,P_m(s_m) \Rightarrow [\,H\,]}{Q_1(s'_1),\ldots,Q_n(s'_n),P_2(s_2),\ldots,P_m(s_m) \Rightarrow [\,H\,]}\ \mathsf{R}$$

R(s, l) return a clause of type (f) when no negative literal is selected in the premise of type (l). Indeed, a clause of type (l) without a selected literal can have one of the following forms: $P_1(y_1),\ldots,P_m(y_m) \Rightarrow P\big(g(y_1,\ldots,y_m)\big)$ which is a particular case of (s), hence the resolution is not possible as seen above, or $P_1(y),\ldots,P_m(y) \Rightarrow P(y)$ where $P_1,\ldots,P_m,P \in \mathcal{P}_0$ $P_1(y),\ldots,P_m(y) \Rightarrow [\,q\,]$ where $q \in \mathcal{Q}$ is a splitting literal or else there is no head. In these two latter cases, the variable in the antecedent is unique, otherwise the clause would be split by ε-splitting. The corresponding resolution steps are the following:

$$\frac{Q_1(x_1),\ldots,Q_n(x_n) \Rightarrow P_1\big(f(\overline{x})\big) \quad P_1(y),\ldots,P_m(y) \Rightarrow P(y)}{Q_1(x_1),\ldots,Q_n(x_n),P_2\big(f(\overline{x})\big),\ldots,P_m\big(f(\overline{x})\big) \Rightarrow P\big(f(\overline{x})\big)}\ \mathsf{R}$$

$$\frac{Q_1(x_1),\ldots,Q_n(x_n) \Rightarrow P_1\big(f(\overline{x})\big) \quad P_1(y),\ldots,P_m(y) \Rightarrow [\,q\,]}{Q_1(x_1),\ldots,Q_n(x_n),P_2\big(f(\overline{x})\big),\ldots,P_m\big(f(\overline{x})\big) \Rightarrow [\,q\,]}\ \mathsf{R}$$

R(s, f) returns a clause of type (f) when one non-splitting literal at least is selected in the premise of type (f) (the tuples $x_1,\ldots,x_n$ and $y_1,\ldots,y_m$ are denoted resp. $\overline{x}$ and $\overline{y}$ below):

$$\frac{\begin{array}{c} Q_1(x_1),\ldots,Q_n(x_n) \Rightarrow P'_1\big(f(\overline{x})\big) \\ P_1(y_{i_1}),\ldots,P_k(y_{i_k}),\underline{P'_1(f(\overline{y}))},\ldots,\underline{P'_m(f(\overline{y}))} \Rightarrow [\,H\,] \end{array}}{P_1(y_{i_1}),\ldots,P_k(y_{i_k}),Q_1(y_1),\ldots,Q_n(y_n),\underline{P'_2(f(\overline{y}))},\ldots,\underline{P'_m(f(\overline{y}))} \Rightarrow [\,H\,]}\ \mathsf{R}$$

R(s, f) returns a clause of type (f) when no literal is selected in the premise of type (f); it is a subcase of R(s, l) (with (l) unselected) above.

R(l, _), R(f, _): no resolution step can involve as first premise a clause of type (l) or (f) which is neither of type (s) nor of type (sp). Indeed, as we have seen above, a clause of this kind and without a selected literal must have the form: $P_1(y),\ldots,P_m(y) \Rightarrow P(y)$ or $P_1(y),\ldots,P_m(y) \Rightarrow [\,q\,]$ In both cases, the head of the clause cannot be strictly maximal w.r.t. $\succ_{lpo}$, contradicting the condition (iii) of LP.

R(sp, l) returns a clause of type (l).

$$\frac{\Rightarrow q_1 \quad \underline{q_1},\ldots,\underline{q_k},Q_1(s_1),\ldots,Q_n(s_n) \Rightarrow [\,H\,]}{\underline{q_2},\ldots,\underline{q_k},Q_1(s_1),\ldots,Q_n(s_n) \Rightarrow [\,H\,]}\ \mathsf{R}$$

Appendix E Relating Reduction Automata and TADE and TAD

Let us consider a reduction automaton $\mathcal{A}$, as defined in Section 5, with equality constraints only, and assume moreover that the transitions of $\mathcal{A}$ fulfill the restrictions on predicates introduced in the definition of TAD. More precisely, it means that for every clause (red) of $\mathcal{A}$ of the form $Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q(f(x_1, \ldots, x_n))[\![c]\!]$:

- c contains no disequality constraint,
- if c contains equality constraints, then Q is a test predicate, and for all $i \leq n$ such that Q_i is a test predicate, $Q \succ Q_i$,
- if c is empty, either $Q_1, \ldots, Q_n, Q \in \mathcal{P}_0$ or Q is a test predicate and at most one of $Q_1, \ldots, Q_n$ is equal to Q, and the others belong to $\mathcal{P}_0$.

We show how to construct a TADE $\mathcal{B}$ of the same size as $\mathcal{A}$ and which recognizes the same language. Let us first consider a $\succ$-convergent sublinear–collapsing theory suitable for that purpose. Let a be the maximal arity of a function symbol of $\mathcal{F}$ and let us add new function symbols $\pi_1, \ldots, \pi_a$ to $\mathcal{F}$. Consider the rewrite system $\mathcal{R}$ containing all rules of the form $\pi_i(f(x_1, \ldots, x_n)) \to x_i$ for $f \in \mathcal{F}$, $i \leq n$. This system is convergent sublinear and collapsing. We add to $\mathcal{B}$ every clause $\Rightarrow \ell = x$ such that $\ell \to x \in \mathcal{R}$.
Every clause (red) of $\mathcal{A}$ as above with an empty constraint c has actually the type t, and is added to $\mathcal{B}$.
To a clause (red) of $\mathcal{A}$ as above with $c = x_{i_1}|_{p_1} = x_{i'_1}|_{p'_1}, \ldots, x_{i_k}|_{p_k} = x_{i'_k}|_{p'_k}$ we associate the following clause of type (d):

$$Q_1(x_1), \ldots, Q_n(x_n), \pi_{p_1}(x_{i_1}) = \pi_{p'_1}(x_{i'_1}), \ldots, \pi_{p_k}(x_{i_k}) = \pi_{p'_k}(x_{i'_k}), x = f(x_1, \ldots, x_n) \Rightarrow Q(x)$$

where $\pi_p(x)$ denotes $\pi_{p_1}(\ldots \pi_{p_k}(x))$ given a position $p = p_1 \ldots p_k$. For every state Q of $\mathcal{A}$, $L(\mathcal{B}, Q) = L(\mathcal{A}, Q)$.

Note that a reduction automaton of the above kind can also be transformed into an equivalent TAD, but at the cost of an exponential explosion, in order to fill with function symbols the positions prefix of p and p' associated to each constraint $x_i|_p = x_{i'}|_{p'}$.

Appendix F Proof of Lemma 2

Lemma 2. We consider terms s, t and a variable x such that $vars(s) \cap vars(t) = \emptyset$, s is not a variable, t is linear $x \in vars(t)$, $x \neq t$, and σ is an mgu of s and t. Then $m_5(x\sigma)$ is strictly less than $m_5(s)$. *Proof.* If s is linear then $x\sigma$ has at most the same number of variables than s and has size smaller than s. Assume now that s is not linear. By applying eagerly Decompose, Orient (and Trivial) rules of unification algorithm to the system $\{s = t\}$ we get the following equivalent system: $\mathcal{X} \cup \mathcal{Y}$ where $\mathcal{X} = \{x_1 = s_1, \ldots, x_n = s_n\}$ and $\mathcal{Y} = \{y_1 = t_1, \ldots, y_m = t_m\}$ with $\{x_1, \ldots, x_n\} \subseteq vars(t)$ and $\{y_1, \ldots, y_m\} \subseteq vars(s)$. The variables $x_1, \ldots, x_n$ have a unique occurrence in the whole system. Let us note too that every variable in t_i's is in $vars(t)$ and therefore occurs only once in the system. As a consequence if we consider the system of equations $\mathcal{T} = \{t_i = t_j \mid y_i = y_j, 1 \leq i, j \leq m\}$. We can check that it has a most general solution σ with support D included in $vars(t) \setminus \{x_1, \ldots, x_n\}$ and for each variable z in this support, $z\sigma$ is a subterm of t and the variables in $z\sigma$ occurs only once in $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{T}$. Applying some replacements the initial system gets equivalent to

$$\mathcal{X} \cup \{y_1 = t_1\sigma, \ldots, y_m = t_m\sigma\} \cup \sigma$$

where by abuse of notation σ is identified with the subsystem $\{z = z\sigma \mid z \in D\}$.

Then from this step all possible Replacement rule applications are performed using some equations of type $y_i = t_i\sigma$.

If x does not occur in a left-hand side in $\mathcal{X}$, then $mvar(x\sigma)$ is a multiset of 1 and therefore strictly less than $mvar(s)$.

Assume now wlog that x is the variable x_1. If no Replacement is applied on s_1, since s_1 is a strict subterm of s, $m_5(x\sigma) < m_5(s)$. If a nonempty sequence of Replacements is applied to s_1, then we get a sequence of right-hand sides: $s_1^1, \ldots, s_1^q$ and we can show by induction that $mvar(s_i^j) < mvar(s)$:

Applying the replacement $y_{j+1} = t_{j+1}$ on s_1^j has effect to replace $|s_1^j|_{y_{j+1}}$ occurrences of variable y_{j+1} by $|s_1^j|_{y_{j+1}}$ occurrences of each of the (linear) variables from t_{j+1}. Since $|s_1|_{y_{j+1}} = |s_1^j|_{y_{j+1}} < |s|_{y_{j+1}}$, then $mvar(t_{j+1}) < mvar(s)$.

□

Appendix G Proof of Proposition 6 (Fact 2)

We detail below the case analysis described in Figure 2.

$\mathsf{Eq}(\mathsf{d})$: equality resolution (Eq) is possible in the equations in clauses of type (d) (each of these equations is selected) and every such application of (Eq) return a clause of the following type (d') with smaller m_2.

$$Q_1(x_1), \ldots, Q_n(x_n), u_1 = v_1, \ldots, u_k = v_k \Rightarrow Q(x)[\![\theta]\!] \qquad (\mathsf{d}')$$

where where $n, k \geq 0$, $x_1, \ldots, x_n$, x are distinct variables, $u_1, v_1, \ldots, u_k, v_k \in \mathcal{T}(\mathcal{F}, \{x_1, \ldots, x_n, x\})$, and $Q > Q_1, \ldots, Q_n$. The type (d') with $k = 0$ (no equation) is just a subcase of (d_+). Recall that the clauses obtained from clauses of type (d) which contain no more equations have type (d_+).

$\mathsf{Eq}(\mathsf{d}')$ also returns a clause of type (d') smaller than the premise.

$\mathsf{RP}(\mathsf{eq}, \mathsf{eq})$ returns a clause of type (eq) which is deleted after simplification by rewriting by $\mathcal{R}$, because by hypothesis, the equational theory of $\mathcal{A}$ is presented as a convergent TRS (hence all critical pairs can be joined).

$\mathsf{RP}(\mathsf{eq}, \mathsf{t})$ returns a clause expanded into type (l_+), smaller than both premises.

$$\frac{\Rightarrow f(\ell_1, \ldots, \ell_n) \to r \quad Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q\big(f(x_1, \ldots, x_n)\big)}{Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q(x)[\![x_1 = \ell_1, \ldots, x_n = \ell_n, x = r]\!]}\ \mathsf{R}$$

Note that no further applications of RP or LP other than resolution is possible into such a clause, because of the basic strategy.

$\mathsf{LP}(\mathsf{eq}, \mathsf{t})$ is not possible with the basic strategy.

$\mathsf{LP}(\mathsf{eq}, \mathsf{d})$ and $\mathsf{LP}(\mathsf{eq}, \mathsf{d}')$ (into the equations selected by sel_2) return constrained clauses of type (d') smaller than both premises. Indeed, every such step suppresses some symbols in the equations hence makes the measure m_3 decrease.

Therefore, the calculus saturates on clauses of type (d) with equations, and terminates either with clauses of type (d_+) (without equations) or with clauses of type (d') with equations which cannot be involved in any paramodulation step. Note that the clauses of type (d_+) can only be involved in resolution steps.

$\mathsf{LP}(\mathsf{eq}, \mathsf{g}_+)$ return a clause of type (g_+) smaller than both premises.

$\mathsf{R}(\_, \mathsf{t})$: no resolution step is possible into a clause of type (t) because of the maximality condition (v) in LP (see the proof of Proposition 4).

$\mathsf{R}(\mathsf{t}, \mathsf{l}_+)$ returns a clause of type (l_+) smaller than both premises, when one non-splitting literal of the second premise (l_+) is selected, i.e. when $\ell_1 = f(\ell'_1, \ldots, \ell'_n)$.

$$\frac{Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow Q'_1\big(f(x_1, \ldots, x_n)\big) \quad \underline{Q'_1(\ell_1)}, \ldots, Q'_m(\ell_m) \Rightarrow Q'(r)}{Q_1(\ell'_1), \ldots, Q_n(\ell'_n), Q'_2(\ell_2), \ldots, Q'_m(\ell_m) \Rightarrow Q'(r)}\ \mathsf{R}$$

Note that with the definition of (t), the multiset of test predicates in (l_+) is unchanged or reduced during this resolution step.

$\mathsf{R}(\mathsf{t}, \mathsf{l}_+)$ returns a clause of type (df) when the premise (l_+) has no selected negative literals. Indeed, such a clause must have the form $Q_1(x), \ldots, Q_n(x) \Rightarrow Q(x)$ where x is a variable, otherwise, it would be split by ε-splitting. Moreover, Q is distinct from $Q_1, \ldots, Q_n$, otherwise the clause would be a tautology (hence if Q is a splitting predicate, it cannot occur amongst $Q_1, \ldots, Q_n$).

$\mathsf{R}(\mathsf{t}, \mathsf{d}_+)$ returns a clause of type (d_+) smaller than both premises when a negative literal $P_1(s_1)$ is selected in the premise (d_+) i.e. when $s_1 = f(s'_1, \ldots, s'_n)$ (the tuple $x_1, \ldots, x_n$ is denoted $\overline{x}$ below):

$$\frac{Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow P_1\big(f(\overline{x})\big) \quad \underline{P_1(s_1)}, \ldots, P_m(s_m) \Rightarrow P^*(s)}{Q_1(s'_1), \ldots, Q_n(s'_n), P_2(s_2), \ldots, P_m(s_m) \Rightarrow P^*(s)}\ \mathsf{R}$$

Note that with the restriction in the definition of (t) concerning the test predicates, the multiset of test predicates in (d_+) is unchanged or reduced during this resolution step.

$\mathsf{R}(\mathsf{t},\mathsf{d}_+)$ returns a clause of type (df) when no negative literal is selected in the second premise (d_+). Indeed, this latter premise must have the form $P_1(x_1),\ldots,P_m(x_m) \Rightarrow P(s)$. Moreover, if s is not a variable, then $x_1\ldots,x_m$ occur in s (otherwise the clause would be split) and $P(s) \succ_{lpo} P_1(x_1),\ldots,P_m(x_m)$, which contradicts the condition (v) of LP. Hence, s is a variable and the premise (d_+) has the form $P_1(x),\ldots,P_m(x) \Rightarrow P(x)$.

$$\frac{Q_1(x_1),\ldots,Q_n(x_n) \Rightarrow P_1\big(f(\overline{x})\big) \quad P_1(x),\ldots,P_m(x) \Rightarrow P^*(x)}{Q_1(x_1),\ldots,Q_n(x_n),Q_2\big(f(\overline{x})\big),\ldots,P_m\big(f(\overline{x})\big) \Rightarrow P^*\big(f(\overline{x})\big)}\,\mathsf{R}$$

$\mathsf{R}(\mathsf{t},\mathsf{df})$ returns a clause of type (df) when the premise (df) has at least one negative literal selected (the tuples of variables $x_1,\ldots,x_n$ and $y_1,\ldots,y_n$ are respectively denoted $\overline{x}$ and $\overline{y}$ below):

$$\frac{\begin{array}{c}Q_1(x_1),\ldots,Q_n(x_n) \Rightarrow Q'_1\big(f(\overline{x})\big)\\ P_1(y_{i_1}),\ldots,P_k(y_{i_k}),\underline{Q'_1\big(f(\overline{y})\big)},\ldots,\underline{Q'_m\big(f(\overline{y})\big)} \Rightarrow Q(r)\end{array}}{P_1(y_{i_1}),\ldots,P_k(y_{i_k}),Q_1(y_1),\ldots,Q_n(y_n),\underline{Q'_2\big(f(\overline{y})\big)},\ldots,\underline{Q'_m\big(f(\overline{y})\big)} \Rightarrow Q(r)}\,\mathsf{R}$$

where r is either y_i or $f(\overline{y})$.

$\mathsf{R}(\mathsf{t},\mathsf{df})$ returns a clause of type (df) when no negative literal is selected in the second premise (df): this case is similar to $\mathsf{R}(\mathsf{t},\mathsf{d}_+)$ when (d_+) is not selected, which has been treated above.

$\mathsf{R}(\mathsf{t},\mathsf{g}_+)$ returns a clause of type (g_+) smaller than both premises when a negative literal $P_1(s_1)$ in the premise (g_+) is selected (i.e. when $s_1 = f(s'_1,\ldots,s'_n)$):

$$\frac{Q_1(x_1),\ldots,Q_n(x_n) \Rightarrow P_1\big(f(\overline{x})\big) \quad \underline{P_1(s_1)},\ldots,P_m(s_m) \Rightarrow [q]}{Q_1(s'_1),\ldots,Q_n(s'_n),P_2(s_2),\ldots,P_m(s_m) \Rightarrow [q]}\,\mathsf{R}$$

$\mathsf{R}(\mathsf{t},\mathsf{g}_+)$ returns a clause of type (gf) when no negative literal is selected in the premise (g_+). In this case, by ε-splitting, the premise (g_+) contains only one variable x:

$$\frac{Q_1(x_1),\ldots,Q_n(x_n) \Rightarrow P_1\big(f(\overline{x})\big) \quad P_1(x),\ldots,P_m(x) \Rightarrow [q]}{Q_1(x_1),\ldots,Q_n(x_n),P_2(f(\overline{x})),\ldots,P_m(f(\overline{x})) \Rightarrow [q]}\,\mathsf{R}$$

$\mathsf{R}(\mathsf{t},\mathsf{gf})$ returns a clause of type (gf) when the premise (gf) has at least one literal selected:

$$\frac{\begin{array}{c}Q_1(x_1),\ldots,Q_n(x_n) \Rightarrow Q'_1\big(f(\overline{x})\big)\\ P_1(y_{i_1}),\ldots,P_k(y_{i_k}),\underline{Q'_1\big(f(\overline{y})\big)},\ldots,\underline{Q'_m\big(f(\overline{y})\big)} \Rightarrow [q]\end{array}}{P_1(y_{i_1}),\ldots,P_k(y_{i_k}),Q_1(y_1),\ldots,Q_n(y_n),\underline{Q'_2\big(f(\overline{y})\big)},\ldots,\underline{Q'_m\big(f(\overline{y})\big)} \Rightarrow [q]}\,\mathsf{R}$$

$\mathsf{R}(\mathsf{t},\mathsf{gf})$ returns a clause of type (gf) when the premise (gf) has no negative literal selected. This is a subcase of the above step $\mathsf{R}(\mathsf{t},\mathsf{g}_+)$ with (g_+) unselected.

$\mathsf{R}(\mathsf{l}_+,\_)$ is not possible. Indeed, let us consider a clause (l_+) without selected negative literals. It must have the (expanded) form $Q_1(x_1),\ldots,Q_n(x_n) \Rightarrow Q(r)$ where $x_1,\ldots,x_n$ are variables and r is either a ground term or a variable. If r is ground, the clause would be split by ε-splitting. If r is a variable, we must have $x_1 = \ldots = x_n = r$ (because of the ε-splitting) and the resolution is not possible because the head of the clause cannot be strictly maximal, contradicting the condition (iii) in LP.

$\mathsf{R}(\mathsf{d}_+,\mathsf{l}_+)$ returns a clause of type (d_+) smaller than both premises.

$$\frac{P_1(x_1),\ldots,P_m(x_m) \Rightarrow Q_1^*(s) \quad Q_1^*(\ell_1),Q_2(\ell_2),\ldots,Q_n(\ell_n) \Rightarrow Q_1^*(r)}{P_1(x_1\theta),\ldots,P_n(x_n\theta),Q_2(\ell_2\theta),\ldots,Q_n(\ell_m\theta) \Rightarrow Q_1^*(r\theta)}\,\mathsf{R}$$

where $\theta = mgu(s,\ell_1)$. Note that all the terms in the antecedents of the premise (d_+) are variables. Otherwise, some literal in this clause would be selected.

By definition of (l_+), $Q_2,\ldots,Q_n$ are not test predicates and for every P_i $(i \leq m)$ which is a test predicate $Q_1^* \succ P_i$. Hence m_4 is strictly smaller for the conclusion (d_+) than for the premise (l_+).

If s is a variable, then, by splitting, we have $x_1 = \ldots, = x_n = s$. This would make the application of LP impossible, because of the ordering condition (iii) of this rule, and definition of the ordering $\succ_{lpo}$.

Hence, we assume that s is not a variable. Moreover, all the variables $x_1, \ldots, x_n$ occur in s, otherwise, the clause (d_+) would be split.

If r is a ground term, or if r is a variable which does not occur in ℓ_1, then $r\theta = r$.

If r is a variable which occurs in ℓ_1, then $r \neq \ell_1$, otherwise the premise (l_+) is a tautology. We can apply Lemma 2 to s, ℓ_1 (which is linear by hypothesis) and r, and if follows that $m_5(s) > m_5(r\theta)$.

In all the above cases, m_5 is strictly smaller for the conclusion than for the first premise (d_+).

$\mathsf{R}(\mathsf{d}_+, \mathsf{d}_+)$ and $\mathsf{R}(\mathsf{d}_+, \mathsf{df})$ both return a clause of type (d_+) smaller than both premises.

$$\frac{Q_1(x_1), \ldots, Q_n(x_n) \Rightarrow P_1(s) \quad P_1(t_1), \ldots, P_m(t_m) \Rightarrow P(t)}{Q_1(x_1\theta), \ldots, Q_n(x_n\theta), P_2(t_2\theta), \ldots, P_m(t_m\theta) \Rightarrow P(t\theta)}\ \mathsf{R}$$

where $\theta = mgu(s, t_1)$.

$\mathsf{R}(\mathsf{d}_+, \mathsf{g}_+)$ returns a clause of type (g_+) smaller than both premises:

$$\frac{Q_1(s_1), \ldots, Q_n(s_n) \Rightarrow P_1(s) \quad P_1(t_1), \ldots, P_m(t_m) \Rightarrow [q]}{Q_1(s_1\theta), \ldots, Q_n(s_n\theta), P_2(t_2\theta), \ldots, P_m(t_m\theta) \Rightarrow [q]}\ \mathsf{R}$$

where $\theta = mgu(s, t_1)$ and $P_1 > Q_1, \ldots, Q_n$.

$\mathsf{R}(\mathsf{d}_+, \mathsf{gf})$ is included in $\mathsf{R}(\mathsf{d}_+, \mathsf{g}_+)$.

$\mathsf{R}(\mathsf{df}, \mathsf{l}_+)$, $\mathsf{R}(\mathsf{df}, \mathsf{d}_+)$, $\mathsf{R}(\mathsf{df}, \mathsf{df})$, and $\mathsf{R}(\mathsf{df}, \mathsf{g}_+)$: there are two cases of clause (df) without selected negative literal. Either such a clause has type (t), and the resolution steps have been treated above, or it has the form $Q_1(x), \ldots, Q_n(x) \Rightarrow Q(x)$, after ε-splitting. In the latter case, Q must be a test predicate. Indeed, otherwise, $Q_1, \ldots, Q_n, Q$ all belong to $\mathcal{P}_0$ and it contradicts the condition (iii) of LP. Hence, the above clause has type (d_+) and the resolution cases are are subcases of $\mathsf{R}(\mathsf{d}_+, \mathsf{l}_+)$, $\mathsf{R}(\mathsf{d}_+, \mathsf{d}_+)$, $\mathsf{R}(\mathsf{d}_+, \mathsf{df})$, and $\mathsf{R}(\mathsf{d}_+, \mathsf{g}_+)$ respectively.

$\mathsf{R}(\mathsf{sp}, \mathsf{l}_+)$ returns a clause of type (l_+) smaller than both premises (note that $m_1(\mathsf{sp}) = \infty$).

$$\frac{\Rightarrow q_1 \quad \underline{q_1}, \ldots, \underline{q_k}, Q_1(\ell_1), \ldots, Q_n(\ell_n) \Rightarrow Q(r)}{\underline{q_2}, \ldots, \underline{q_k}, Q_1(\ell_1), \ldots, Q_n(\ell_n) \Rightarrow Q(r)}\ \mathsf{R}$$

$\mathsf{R}(\mathsf{sp}, \mathsf{d}_+)$ returns a clause of type (d_+) smaller than both premises.

$$\frac{\Rightarrow q_1 \quad \underline{q_1}, \ldots, \underline{q_k}, Q_1(s_1), \ldots, Q_n(s_n) \Rightarrow Q(s)}{\underline{q_2}, \ldots, \underline{q_k}, Q_1(s_1), \ldots, Q_n(s_n) \Rightarrow Q(s)}\ \mathsf{R}$$

$\mathsf{R}(\mathsf{sp}, \mathsf{g}_+)$ returns a clause of type (sp) or a clause of type (g_+) smaller than both premises.

$$\frac{\Rightarrow q_1 \quad \underline{q_1}, [\ \underline{q_2}, \ldots, \underline{q_k}, P_1(s_1), \ldots, P_m(s_m)\ ] \Rightarrow [\ q\ ]}{[\ \underline{q_2}, \ldots, \underline{q_k}, P_1(s_1), \ldots, P_m(s_m)\ ] \Rightarrow [\ q\ ]}\ \mathsf{R}$$

Note that the conclusion is indeed smaller than the premise (sp) because $m_1(\mathsf{sp}) = \infty$.

$\varepsilon\mathsf{split}(\mathsf{l}_+)$: the ε-splitting of such clauses returns a clause of type (gf) and a clause of type (l_+) smaller than the premise (B is an ε-block):

$$\frac{q_1, \ldots, q_k, B, Q_1(\ell_1), \ldots, Q_n(\ell_n) \Rightarrow Q(r)}{B \Rightarrow q_B \quad q_1, \ldots, q_k, q_B, Q_1(\ell_1), \ldots, Q_n(\ell_n) \Rightarrow Q(r)}\ \varepsilon\mathsf{split}$$

εsplit(d_+): the ε-splitting of a clause of type (d_+) returns a clause of type (gf) and a clause of type (d_+) smaller than the premise (B is an ε-block):

$$\frac{q_1, \ldots, q_k, B, Q_1(s_1), \ldots, Q_n(s_n) \Rightarrow Q(s)}{B \Rightarrow q_B \quad q_1, \ldots, q_k, q_B, Q_1(s_1), \ldots, Q_n(s_n) \Rightarrow Q(s)}\ \varepsilon\mathsf{split}$$

εsplit(g_+): the ε-splitting of a clause of type (g_+) returns a clause of type (gf) and a clause of type (g_+) smaller than the premise (B is an ε-block):

$$\frac{q_1, \ldots, q_k, B, P_1(s_1), \ldots, P_m(s_m) \Rightarrow [q]}{B \Rightarrow q_B \quad q_1, \ldots, q_k, q_B, P_1(s_1), \ldots, P_m(s_m) \Rightarrow [q]}\ \varepsilon\mathsf{split}$$

Unité de recherche INRIA Futurs
Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

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