Tree automata with equality constraints modulo equational theories

Florent Jacquemard, Michael Rusinowitch, Laurent Vigneron

Tree automata with equality constraints modulo equational theories ⋆

Florent Jacquemard [a], Michael Rusinowitch [b] and Laurent Vigneron [c]
[a] *INRIA Futurs & LSV, UMR 8643*
[b] *LORIA & INRIA Lorraine, UMR 7503*
[c] *LORIA & Univ. Nancy 2, UMR 7503*

Abstract

This paper presents new classes of tree automata combining automata with equality test and automata modulo equational theories. We believe that these classes have a good potential for application in *e.g.* software verification. 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 (extending the emptiness problem) is decidable by proving that the saturation of tree automata presentations with suitable paramodulation strategies terminates. Alternatively our results can be viewed as new decidable classes of first-order formula.

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

1. Introduction

Combining tree automata and term rewriting systems (TRS) has been successful in domains like automated theorem proving [1] and verification of infinite state systems *e.g.* [2–4].

A problem with such approaches is to extend the decidability results on tree automata languages to equivalence classes of terms modulo an equational theory. Some authors, *e.g.* [5,6], have investigated the problem of emptiness decision for tree automata modulo specific equational theories, *e.g.* A, AC, ACU... Moreover, it is also shown in [6] that emptiness is decidable for any linear equational theory, and results about regularity preservation under rewriting have been established for several general classes of TRS (see *e.g.* [7] § 2.3).

Another important difficulty stems from the non linear variables (variables with multiple occurrences) in the rewrite rules, which impose in general some over-approximations of the rewrite relation. Tree automata with constraints have been proposed earlier in order to deal with non-linear rewrite systems (see [1]). They are an extension of classical tree recognizers where syntactic equality and disequality tests between subterms are performed during the automata computations. The emptiness of the recognized language is undecidable

⋆ This work has been partially supported by the research projects RNTL PROUVÉ (No 03 V 360) and ACI–SI SATIN and ROSSIGNOL.

Email addresses: florent.jacquemard@inria.fr (Florent Jacquemard), michael.rusinowitch@loria.fr (Michael Rusinowitch), laurent.vigneron@loria.fr (Laurent Vigneron).

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

Fig. 1. Presentation of the decision results in the paper.

without restriction, and two remarkable subclasses with decidable emptiness problem are tree automata with equality and disequality constraints restricted to brother positions of [8] and the *reduction automata* of [9]. This second class captures in particular languages of terms (ir)reducible by non linear rewrite systems.

Following [10], it is classical to represent tree automata by Horn clause sets. In this setting, a recognized language is defined as a least Herbrand model and it is possible to use classical first-order theorem proving techniques in order to establish decision results [5,11].

In this paper, we follow this approach in order to unify the two problems mentioned above: we show how techniques of basic ordered paramodulation with selection and a variant of splitting without backtracking solve some decision problems on languages of tree automata with equality constraints, transformed by rewriting. More precisely, we show that the so called *Generalized Intersection Problem*, GIP (whether there exists a ground instance of a given term tuple in a given language tuple) is decidable by saturation with a standard calculus presented in Section 3. Note that GIP generalizes the emptiness problem. Alternatively our results can be viewed as new decidable classes of first-order formula. Both classes of standard tree automata (TA) and tree automata with equality constraints (TAC) generalizing those of [9], where the equality tests are presented by arbitrary equations, are studied in these settings, as well as their respective generalisations (TAE and TACE) modulo an equational theory $\mathcal{E}$ presented as a convergent term rewriting system (monadic TRS in the case of TAE and restricted collapsing TRS in the case of TACE).

Figure 1 summarizes the presentation of the decision results in the paper. The last result (lower right corner of the table in Figure 1) is to our knowledge one of the first decision results (after [12]) 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 [9,1], we prove also that this problem is undecidable for non-deterministic reduction automata (see Section 6.1). Therefore, we have introduced for the definition of TAC a refinement on the restriction for the automata of [9] in order to make emptiness and, more generally, GIP 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 GIP (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.

As illustrated by two examples of authentication protocols (one with recursion) 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).

Related work. A comparison with the reduction automata of [9] is detailed in Sections 6 and 7. The closely related works [3,11] propose a different extension $\mathcal{H}_1$ of standard TA defined as Horn clause sets for which satisfiability is decidable. In the version [11] of $\mathcal{H}_1$ 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 TAC and $\mathcal{H}_1$ contains the other. However, $\mathcal{H}_1$ becomes undecidable when allowing variable duplication in the heads. Our TAC 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 denote $vars(t)$ as 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. We assume standard definitions and notations for term rewriting [13]. 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 $\mathcal{R}$, denoted by $t \to_{\mathcal{R}} s$, 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 TRS $\mathcal{R}$ is *confluent* if for all terms t, s_1, s_2 such that $s_1 \overset{*}{\leftarrow}_{\mathcal{R}} t \overset{*}{\to}_{\mathcal{R}} s_2$ ($\overset{*}{\to}_{\mathcal{R}}$ denotes the reflexive and transitive closure of the rewrite relation $\to_{\mathcal{R}}$ defined by $\mathcal{R}$) there exists a term t' such that $s_1 \overset{*}{\to}_{\mathcal{R}} t' \overset{*}{\leftarrow}_{\mathcal{R}} s_2$. A TRS is *convergent* if it is both terminating and confluent.

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 later use a partition $\mathcal{P} \setminus \{=\} = \mathcal{P}_0 \uplus \mathcal{P}_1$, where $\mathcal{P}_0$ and $\mathcal{P}_1$ are sets of predicate symbols. 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 the 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 the *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 $P(s) = \mathit{true}$, resp. $q = \mathit{true}$, where *true* is a distinguished function symbol (in $\mathcal{F}$). An atom of the latter form is called *non-equational* 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 we are given a *precedence* ordering $\succeq$ on $\mathcal{F} \cup \mathcal{P} \cup \mathcal{Q}$, and denote by $\sim$ 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* [13] 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 [14] 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. Tree automata are finite state recognizers of ground terms. We consider here a definition à la Frühwirth et al [10] of tree automata 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 by $L(\mathcal{A}, Q)$, is the set of terms $t \in \mathcal{T}(\mathcal{F})$ such that $Q(t)$ is a logical consequence of $\mathcal{A}$.

General Intersection Problem (GIP). We focus on one decision problem, GIP, which generalizes many important problems concerning tree automata (in particular membership and emptiness decision).

INSTANCE: a tree automaton $\mathcal{A}$, some states $Q_1, \ldots, Q_n$ of $\mathcal{A}$ and some terms $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$,

QUESTION: is there a substitution σ grounding for $t_1, \ldots, t_n$ such that for all $i \geq n$, $t_i\sigma \in L(\mathcal{A}, Q_i)$?

When $t_1 = \ldots = t_n = x$ (a variable), GIP is equivalent to the problem of non-emptiness of intersection of tree automata, which is known to be EXPTIME-complete [15]. An inclusion problem $L(\mathcal{A}, P) \subseteq L(\mathcal{B}, Q)$ is a particular case of GIP when $\mathcal{B}$ belong to a class of TA closed under complementation: in this case, inclusion can be expressed as GIP for P, $\overline{Q}$ and $t_1 = t_2 = x$, where $\overline{Q}$ is a state of a complement tree automata $\overline{\mathcal{B}}$ such that $L(\overline{\mathcal{B}}, \overline{Q})$ is the complement of $L(\mathcal{B}, Q)$ in $\mathcal{T}(\mathcal{F})$.

The *General Membership Problem* (GMP, [16]) is the particular case of GIP where $n = 1$. This problem was shown EXPTIME-complete in [17] for standard tree automata. When t_1 is a ground term, GMP is equivalent to a *membership* problem for $\mathcal{A}$: $t \in L(\mathcal{A}, Q)$? When t is a variable, GMP is equivalent to a *non-emptiness* problem for $\mathcal{A}$: $L(\mathcal{A}, Q) \neq \emptyset$?

Lemma 1 *GIP is satisfied by $\mathcal{A}$, Q and t iff $\mathcal{A} \cup \{Q_1(t_1), \ldots, Q_n(t_n) \Rightarrow \}$ is inconsistent.*

3. Basic Ordered Paramodulation with Selection

We shall establish the decidability of GIP 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 *selection function* which assigns to each clause a set of selected 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* [14,18].

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

if x is fresh, and (i) $\ell' \notin \mathcal{X}$, (ii) no literal is selected in Γ and Γ', (iii) and (v) hold.

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

if x is fresh, (i) $\ell' \notin \mathcal{X}$, (ii) no literal is selected in Γ, (iii) holds, (iv) $u = v$ is selected or (v') holds.

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

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

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

The conditions missing above are: (iii) $\ell\sigma \not\preceq r\sigma$ and $\ell\sigma = r\sigma$ is strictly maximal in $\Gamma\sigma, \ell\sigma = r\sigma$, (v) $u\sigma = v\sigma$ is maximal in $\Gamma'\sigma, u\sigma = v\sigma$, where σ is the most general unifier (mgu) of $\theta, \theta', \ell' = \ell, x = r$, (v') $u\sigma = v\sigma$ is maximal in $\Gamma'\sigma, u\sigma = v\sigma, A\sigma$ (σ is as in (v)).

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 the *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*.

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

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

Note that the clause generated by the LP step is deleted, subsumed by the clause generated by the Eq step.

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 [14].

Splitting. We shall use *ε-splitting* [11], a variant of *splitting without backtracking* [19].

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

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 with 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 [14]) 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 2 *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.

Example 3 *The language of the following TA in Q_1 is the set of binary trees with inner nodes labelled by* f *and leaves labelled by* 0 *or* 1*, such that at least a leaf is labeled by* 1*:* $\Rightarrow Q_0(0), \quad \Rightarrow Q_1(1),$

$$Q_0(x_1), Q_0(x_2) \Rightarrow Q_0(\mathsf{f}(x_1, x_2)), \quad Q_1(x_1), Q_0(x_2) \Rightarrow Q_1(\mathsf{f}(x_1, x_2)),$$
$$Q_0(x_1), Q_1(x_2) \Rightarrow Q_1(\mathsf{f}(x_1, x_2)), \quad Q_1(x_1), Q_1(x_2) \Rightarrow Q_1(\mathsf{f}(x_1, x_2))$$

$\diamond$

4.1. *Decision of GIP*

The emptiness and membership problems for TA 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 is EXPTIME-complete [20]. We sketch below a slight variation of a DEXPTIME procedure of [11] in our framework, in order to introduce the principles of the proofs in the next sections. It is based on the function sel_1 which selects in a Horn clause $\Gamma \Rightarrow H[\![\theta]\!]$:
– every splitting negative literal, if any,
– and otherwise every non-equational literal $Q(t)$ of Γ such that $t\theta$ is not a variable.

Proposition 4 ([11]) *Ordered resolution with selection and ε-splitting saturates the union of a TA $\mathcal{A}$ and a goal clause $P_1(t_1), \ldots, P_n(t_n) \Rightarrow$.*

proof. We assume *wlog* that $P_1, \ldots, P_n \in \mathcal{P}_0$, otherwise the problem is trivial by definition of (s). We show that the saturation of a TA and the goal $P_1(t_1), \ldots, P_n(t_n) \Rightarrow$ under ordered resolution wrt $\succ_{lpo}$ and the selection function sel_1, with eager application of the $\varepsilon\mathsf{split}$ rule of Section 1, produce only clauses of one of the following form (gs), for goal-subterm, or (gf), for goal-flat.

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

where $m, k \geq 0$, $s_1, \ldots, s_m$ are subterms of 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).

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

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 (optional in the clause).

The particular subtype of (gf) of positive clauses with a splitting literal as head (i.e. (gf) with $k = m = 0$) is denoted (sp) below:

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

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

Note that the initial goal $P_1(t_1), \ldots, P_n(t_n) \Rightarrow$ belongs to the type (gs) and also belongs to (gf) if t is a variable. The cases of resolution are listed in Appendix A. Since the number of clauses of type (gs) and (gf) is exponential, the saturation terminates. $\square$

Corollary 5 *GIP is decidable for TA.*

4.2. *Undecidable extension.*

Let us call a *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 (even with only one predicate):

Proposition 6 *GMP for sets of clauses of type (s_+) is undecidable.*

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 $q(s,t) \to q'(s',t')$ of the

	MACHINE INSTRUCTION	CLAUSAL REPRESENTATION
T_1	q : ADD 1 TO COUNTER 1; GOTO q'	$\Rightarrow Q(g(q(x,y),h(g(q'(s(x),y),u),u)))$
T_2	q : IF COUNTER 1 $\neq 0$ DEC 1; GOTO q'	$\Rightarrow Q(g(q(s(x),y),h(g(q'(x,y),u),u)))$
T_3	q : IF COUNTER 1 $= 0$; GOTO q'	$\Rightarrow Q(g(q(0,y),h(g(q'(0,y),u),u)))$

Fig. 2. Representation of the transitions of the 2 counter machine as facts.

machine by a fact, as described in Figure 2 for three examples. We will need for this purpose a predicate symbol Q, some binary functions g, h, k and a constant symbol c.

See also Figure 4 for a tree 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 also introduce two auxiliary (s) clauses:

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

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

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 ($\mathsf{h}$) 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 $\Rightarrow Q(g(q_{\mathsf{f}}(x,y),c))$, 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. □

Note that GMP with clauses of type (s) and linear facts is reducible to emptiness for standard TA [1], hence decidable.

5. 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 [7], Section 2.3 for a summary of some recent results). These results often rely on a procedure of *completion* of TA wrt some TRS, which adds new TA transitions without adding new states. As observed in [12], 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 7 *A rewrite rule $\ell \rightarrow r$ 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 8 *The following axiom for integer equality:* $\mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) \rightarrow \mathsf{eq}(x,y)$ *as well as this rule for the elimination of stuttering in lists:* $\mathsf{cons}(x,\mathsf{cons}(x,y)) \rightarrow \mathsf{cons}(x,y)$ *are monadic rewrite rules. Sublinear and collapsing rewrite rules permit to describe cryptographic functions [21], like decryption in a symmetric cryptosystem* $\mathsf{dec}\big(\mathsf{enc}(x,y),y\big) \rightarrow x$ *(the symbols* enc *and* dec *stand for encryption and decryption and the variables x and y correspond respectively to the encrypted plaintext and the encryption key), or, in the case of public (asymmetric) key cryptography:* $\mathsf{adec}\big(\mathsf{aenc}(x,\mathsf{pub}(y)),\mathsf{inv}(\mathsf{pub}(y))\big) \rightarrow x$ *and* $\mathsf{adec}\big(\mathsf{aenc}(x,\mathsf{inv}(\mathsf{pub}(y))),\mathsf{pub}(y)\big) \rightarrow x$ *where* inv *is an idempotent operator, following the rule* $\mathsf{inv}(\mathsf{inv}(y)) \rightarrow y$*, which associates to a public encryption key its corresponding private key (for decryption), and conversely. We will also consider below projections on pairs:* $\mathsf{fst}(\mathsf{pair}(x,y)) \rightarrow x$ *and* $\mathsf{snd}(\mathsf{pair}(x,y)) \rightarrow y$. ◇

We call an *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 $\mathcal{R} = \{\ell \rightarrow r \mid \Rightarrow \ell = r \in \mathcal{E} \text{ and } \ell \succ_{lpo} r\}$ 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 9 *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*).*

Example 10 *The language of the following simple TAE in state* Q_{e} *is the set of expressions equivalent to non-negative even integers:*

$$\Rightarrow \mathsf{p}(\mathsf{s}(x)) = x \qquad \Rightarrow \mathsf{s}(\mathsf{p}(x)) = x$$
$$\Rightarrow Q_{\mathsf{e}}(0) \quad Q_{\mathsf{e}}(x) \Rightarrow Q_{\mathsf{o}}(\mathsf{s}(x)) \qquad Q_{\mathsf{o}}(x) \Rightarrow Q_{\mathsf{e}}(\mathsf{s}(x))$$

If, instead of the above equational theory for successor and predecessor we consider the following monadic equational theory for a partial subtraction on natural numbers: $\mathsf{s}(x) - \mathsf{s}(y) = x - y, x - 0 = x, 0 - x = 0$*, the language is the set of ground terms equivalent to non-negative even integers.* ◇

Proposition 11 *Basic ordered paramodulation with selection and* ε*-splitting. saturates the union of a TAE* $\mathcal{A}$ *modulo a* $\succ$*-convergent monadic equational theory and a goal clause* $P_1(t_1), \ldots, P_n(t_n) \Rightarrow$*.*

proof. We show the termination of the saturation of $\mathcal{A} \cup \{P_1(t_1), \ldots, P_n(t_n) \Rightarrow\}$ under basic ordered paramodulation wrt the ordering $\succ_{lpo}$ and the selection function sel_1 (defined in the proof of Proposition 4) and with eager ε-splitting.

The main difference with the situation of Proposition 4 is that some rule of the equational theory (i.e. a clause of type (eq)) may be applied to a clause of type (s) by right paramodulation RP.

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

Also, LP with an equational clause (eq) is possible into the initial goal clause $P(t) \Rightarrow$. We introduce below a new clause type (l) to characterize the (expansions of) clauses obtained this way, and show by a case analysis that all the clauses obtained during the saturation are of type (l) or of a type (f) which generalizes (gf) (proof of Proposition 4).

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.

The clause type (l) is defined as follows:

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

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 $Q \in \mathcal{P}_0$ where 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 (l).

The following type (f) generalizes the type (gf) defined in the proof of Proposition 4:

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

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 $P_1(t_1), \ldots, P_n(t_n) \Rightarrow$ belongs to type (l) and also belongs to (f) if t is a variable. The different cases of saturation are summarized in Figure 3 and detailed in Appendix B. Since the number of clauses of type (l) and (f) is finite, this proves that the saturation of $\mathcal{A} \cup \{P_1(t_1), \ldots, P_n(t_n) \Rightarrow\}$ under basic ordered paramodulation terminates. □

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)

Fig. 3. Case analysis in the proof of Proposition 11

Note that the expanded form of the above clause $Q_1(\ell_1), \ldots, Q_n(\ell_n) \Rightarrow Q(y)$ is related to the push clauses of two-ways automata [5] or *selecting theories* [22]. We will come back to this remark in Section 7.5 showing how the approach for protocol verification of this last paper can be carry on by TACE.

Corollary 12 *GIP is decidable for TAE modulo a $\succ$-convergent monadic equational theory.*

6. Tree Automata with Syntactic Equational Constraints

6.1. *Reduction Automata*

The original reduction automata (RA) of [9] can be defined as finite sets 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 $x_i|_p \neq x_{i'}|_{p'}$ (disequality 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 $t|_p \neq t'|_{p'}$). Given an RA $\mathcal{A}$ and a state Q of $\mathcal{A}$, the language $L(\mathcal{A}, Q)$ is defined as in page 4 (extending the definition from Horn clause to constrained Horn clauses). The definitions of GIP, GMP and emptiness problems for RA follow.

We prove that the emptiness problem is undecidable for non-deterministic reduction automata, contradicting a claim in [9,1].

Proposition 13 *The emptiness problem is undecidable for non-deterministic RA.*

proof. As in the proof of Proposition 6, 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 6, 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 $\mathcal{A}$ with states Q_0 (for 0) and Q_1 (for strictly positive integers), a universal state $Q_\forall$, some states Q_c, Q_d, Q_{gd}, Q_{hd}, and T for every transition $T : c \to d$ of $\mathcal{M}$, a state Q for chaining the transitions of $\mathcal{M}$ and a final state Q_f.

This reduction automaton $\mathcal{A}$ is such that the language $L(\mathcal{A}, Q_\mathsf{f})$ is the set of 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_{01} is an abbreviation for either Q_0 or Q_1.

The transition $T_1 = c_1 \to d_1$ 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 4 for sake of readability). We use the following states and clauses for the recognition of this term T_1:

– $Q_{01}(x_1), Q_{01}(x_2) \Rightarrow Q_{c_1}(q(x_1, x_2))$ and $Q_1(x_1), Q_{01}(x_2) \Rightarrow Q_{d_1}(q'(x_1, x_2))$ where the states Q_{c_1} Q_{d_1} are respectively associated to the left and right members of the transition,

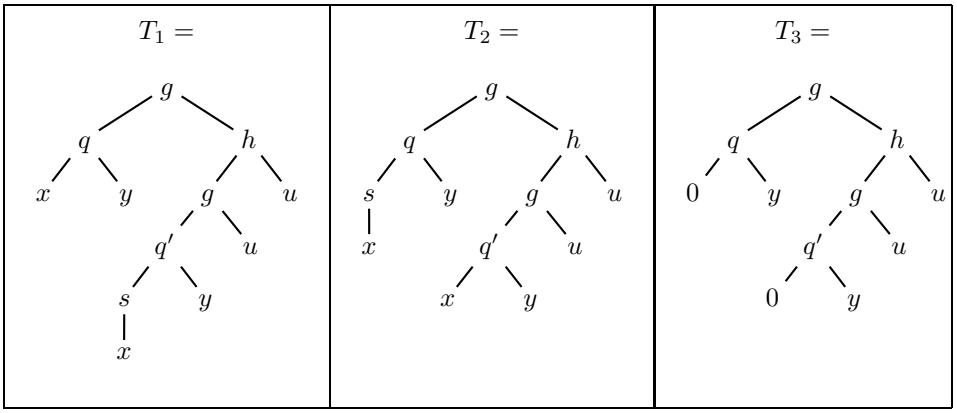

Fig. 4. Representation of the transitions of the 2 counter machine as trees.

– $Q_{d_1}(x_1), Q_\forall(x_2) \Rightarrow Q_{gd_1}(g(x_1, x_2))$,
– $Q_\forall$ 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 4 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 4). We use the following states and clauses for the recognition of this term T_2:
– $Q_1(x_1), Q_{01}(x_2) \Rightarrow Q_{c_2}(q(x_1, x_2))$ and $Q_{01}(x_1), Q_{01}(x_2) \Rightarrow Q_{d_2}(q'(x_1, x_2))$,
– $Q_{d_2}(x_1), Q_\forall(x_2) \Rightarrow Q_{gd_2}(g(x_1, x_2))$, $Q_{gd_2}(x_1), Q_\forall(x_2) \Rightarrow Q_{hd_2}(h(x_1, x_2))$,
– and the last clause (for the recognition of the term T_2, Figure 4) 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 4). We use the following states and clauses for the recognition of this term T_3:
– $Q_0(x_1), Q_{01}(x_2) \Rightarrow Q_{c_3}(q(x_1, x_2))$ and $Q_0(x_1), Q_{01}(x_2) \Rightarrow Q_{d_3}(q'(x_1, x_2))$,
– $Q_{d_3}(x_1), Q_\forall(x_2) \Rightarrow Q_{gd_3}(g(x_1, x_2))$, $Q_{gd_3}(x_1), Q_\forall(x_2) \Rightarrow Q_{hd_3}(h(x_1, x_2))$,
– and the last clause (for the recognition of the term T_3, Figure 4) 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 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 $L(\mathcal{A}, Q)$:

$$\Rightarrow Q_c(c), \qquad Q_{01}(x_1), Q_{01}(x_2) \Rightarrow Q_{c_{\mathsf{f}}}(q_{\mathsf{f}}(x_1, x_2)),$$
$$Q_{c_{\mathsf{f}}}(x_1), Q_c(x_2) \Rightarrow Q_{gc_{\mathsf{f}}}(g(x_1, x_2)), \quad Q_{gc_{\mathsf{f}}}(x_1), Q_c(x_2) \Rightarrow Q(h(x_1, x_2))$$

An example of a computation of $\mathcal{A}$ is described in Figure 5. 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.

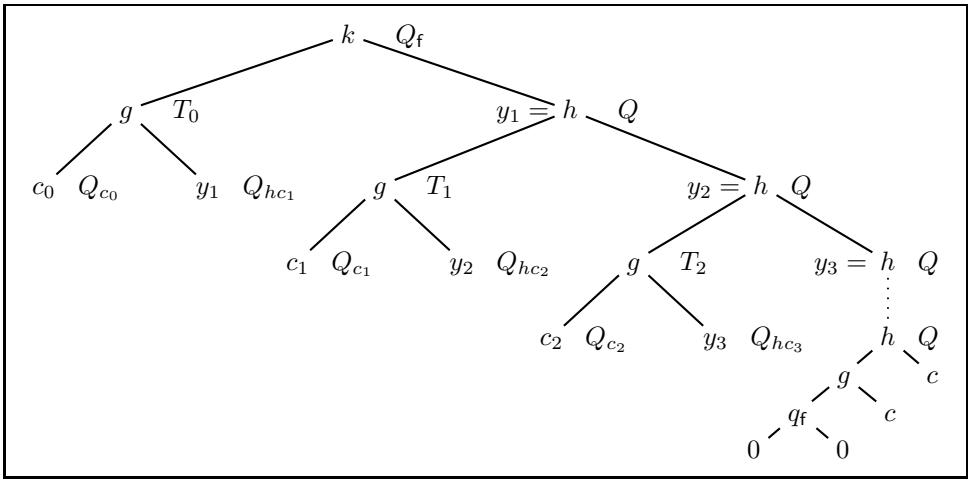

Fig. 5. A computation of $\mathcal{A}$

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 5.

For the if direction, we use the following fact:

Fact 14 *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 14, and with this fact, this ensures that t represents a halting computation of $\mathcal{M}$. □

6.2. *Tree Automata with Equational Constraints*

We propose here the definition of a new class of tree automata where the constraints are generalized (compared to [9]) to equations between arbitrary terms and where the transitions comply to stronger ordering conditions, based on the ordering $\succ$ on states, in order to obtain a decidable GIP. We call below *test predicates* [2] 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 (s) which contain no more test predicates symbols in their antecedents than 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 $\mathcal{P}_0$.

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

Definition 15 *A tree automaton with equational constraints or TAC is a finite set of clauses of type (*t*) or (*d*).*
Note that every TA is a particular case of TAC (without test predicates).
Example 16 *The language of the following TAC in state Q_2 is the set of stuttering lists of natural numbers build with the symbols* cons *and* empty*:*

$$\begin{array}{rr} \Rightarrow Q_0(0) & Q_0(x_1) \Rightarrow Q_0(\mathsf{s}(x_1)) \\ \Rightarrow Q_1(\mathsf{empty}) & Q_0(x_1), Q_1(x_2) \Rightarrow Q_1(\mathsf{cons}(x_1, x_2)) \\ & Q_0(x_1), Q_2(x_2) \Rightarrow Q_2(\mathsf{cons}(x_1, x_2)) \\ Q_0(x_1), Q_1(x_2), x_2 = \mathsf{cons}(x_1, y), x = \mathsf{cons}(x_1, x_2) \Rightarrow Q_2(x) & \end{array}$$

$\diamond$

Proposition 17 *Ordered paramodulation with selection and ε-splitting saturates the union of a TAC $\mathcal{A}$ and a goal clause $P_1(t_1), \ldots, P_n(t_n) \Rightarrow$.*
proof. Let sel_2 be a selection function which generalizes sel_1, by selecting every equational negative literals, if any, and otherwise is defined just like sel_1; this means that sel_2 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 consider saturation under ordered paramodulation wrt $\succ_{lpo}$ with selection by sel_2 and ε-splitting. The principle of the proof of termination (detailed in Appendix C) is to show that, starting with $\mathcal{A} \cup \{P_1(t_1), \ldots, P_n(t_n) \Rightarrow\}$, every step of ordered paramodulation wrt the ordering $\succ_{lpo}$ and the selection function sel_2, with eager ε-splitting. returns either a clause smaller than all its premises (wrt to a well founded ordering $\gg$) or a clause of type (gf). Two key points ensure this result. First, because of the selection strategy, the clauses of $\mathcal{A}$ of type (d) containing equations can only be involved in an equality resolution (Eq). Hence, equations in such clauses will be eliminated first, before these clauses can be involved in resolution. The type of clauses obtained (when all equations have been eliminated) is called (d_+) below and their predicates satisfy the same ordering condition as for (d). Second, thanks to the ordering conditions on predicates for (t) and (d_+), the application of such clauses in resolution makes clauses decrease (wrt $\gg$).

Let us now give a formal definition of the type (d_+) of clauses obtained by equation elimination:

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

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 $Q^* \succ Q_i$.

The transformation, by equation elimination, of clauses of type (d) into clauses of type (d_+) (first keypoint above) is summarized in Figure 6 and detailed in Appendix C.

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 4, 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 $P_1(t_1), \ldots, P_n(t_n) \Rightarrow$ has type (g_+).

The following Fact 18 states formally the above second key point. It is based on the following 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. This measure is the tuple made of the following components:
– $m_1(C) = ind(Q)$ (see page 3 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,

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	$\diagup$								
R	_	d_+	$\diagup$								
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)

Fig. 6. Case analysis in the proof of Proposition 17. $>$ (resp. $\geq$, $=$) means that the measure's component for the premise is strictly greater than (resp. greater or equal to, equal to) the conclusion.

– $m_4(C)$ 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 $\left(>, >, \succ^{mul}, \succ_{lpo}^{mul}\right)^{lex}$, where $>$ denotes the ordering on natural numbers. The proof of Fact 18, based on a case analysis, is summarized in Figure 6 and detailed in Appendix C.

Fact 18 *Starting with $\mathcal{A} \cup \{P_1(t_1), \ldots, P_n(t_n) \Rightarrow\}$, 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 18 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. □

Corollary 19 *GIP is decidable for TAC.*

7. Tree Automata with Equational Constraints Modulo a Theory

It is shown in [12] that the class of languages of terms recognized by tree automata of [8] (tree automata with equality constraints between brother positions) is 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 [9]. 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 TAC 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 20 *A tree automaton with equational constraints modulo an equational theory (TACE) is the union of an equational theory and of a TAC.*

7.1. *Relating RA and TACE*

We show in this section that every reduction automaton with equality constraints only is equivalent to a TACE 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.

Let us consider a reduction automaton $\mathcal{A}$, as defined in Section 6.1, with equality constraints only, and assume moreover that the transitions of $\mathcal{A}$ fulfill the restrictions on predicates introduced in the definition of TAC. 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 TACE $\mathcal{B}$ 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 TAC, 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 $x_i|_p = x_{i'}|_{p'}$.

7.2. *Example: modeling a security protocol*

We illustrate in the following example how TACE can be used to characterize the behaviour of security protocols running in an insecure environment, following a model with explicit destructors [21] specified with the rewrite rules of Example 8. It is known [23] that such model with rewrite rules is more expressive than a standard model of cryptosystems based on free algebras. For instance, the attack mentioned in Section 7.4 cannot be captured by free algebras based approach like *e.g.* [2]. Our representation is such that a state of the protocol is reachable (from an initial state) iff it is in the TACE language.

Example 21 *The protocol of Denning & Sacco [24] permits two agents A and B to exchange a new symmetric key using an asymmetric cryptosystem. The respective behaviour of the agents can be represented by*

the two following clauses of type (d)[3] *:*

$$Q_{0j}(x) \Rightarrow Q_{1j}(\mathsf{pair}(A, \mathsf{aenc}(\mathsf{aenc}(K, \mathsf{inv}(\mathsf{pub}(A))), \mathsf{pub}(B)))) \qquad j = 0, 1$$
$$Q_{i0}(x) \Rightarrow Q_{i1}(\mathsf{enc}(S, \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(x))))) \; i = 0, 1$$

The predicate Q_{ij} represents the content of the channel Q when agents A and B are in respective states i, j, which are either 0 (initial state) or 1 (final state). In the first clause, A initiates the protocol, sending B a freshly chosen symmetric key K for further secure communications (A, B, K, S are constant function symbols). This key is K signed, for authentication purpose, with the secret key $\mathsf{inv}(\mathsf{pub}(A))$ of A and encrypted with the public key $\mathsf{pub}(B)$ of B. Moreover, A appends its name at the beginning of the message. In the second clause, B answers with a secret value S encrypted with K, which has been extracted from the received message (using the destructor symbols and the rules of Example 8). Note that in this setting, equations in clauses (d) permit to model conditionals for the agents of protocols.

We add some clauses of type (t) and (d) in order to model the control of an attacker over the public communication channel Q, namely the ability to read, analyze, recompose (by application of any public function f, possibly a destructor symbol) and to resend messages:

$$Q_{00}(x_1), Q_{00}(x_2) \Rightarrow Q_{00}(\mathsf{f}(x_1, x_2)) \qquad Q_{00}(x_1), Q_{01}(x_2) \Rightarrow Q_{01}(\mathsf{f}(x_1, x_2))$$
$$Q_{00}(x_1), Q_{10}(x_2) \Rightarrow Q_{10}(\mathsf{f}(x_1, x_2)) \qquad Q_{00}(x_1), Q_{11}(x_2) \Rightarrow Q_{11}(\mathsf{f}(x_1, x_2))$$
$$\textit{symmetric of the above clauses:} \qquad Q_{01}(x_1), Q_{00}(x_2) \Rightarrow Q_{01}(\mathsf{f}(x_1, x_2)) \ldots$$
$$Q_{01}(x_1), Q_{10}(x_2) \Rightarrow Q_{11}(\mathsf{f}(x_1, x_2)) \qquad Q_{10}(x_1), Q_{01}(x_2) \Rightarrow Q_{11}(\mathsf{f}(x_1, x_2))$$

Note that in the above clauses we allow several combinations of the agent's states in the antecedents, but not every combination. The principle is that if A (resp. 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. This way, we ensure an exact representation (as ground terms) of the executions of an instance of the protocol, whereas many other Horn clauses or tree automata models are approximating [2,3,5]. Note that these conditions fit well with the ordering restrictions on clauses of type (t) and (d). We also add some clauses (t) ensuring that some ground terms are initially known to the attacker, e.g. $\Rightarrow Q_{00}(A)$. ◇

7.3. *GIP and TACE*

Proposition 22 *Basic ordered paramodulation with selection and ε-splitting saturates the union of a TACE $\mathcal{A}$ modulo a $\succ$-convergent sublinear and collapsing equational theory and a goal clause $P_1(t_1), \ldots, P_n(t_n) \Rightarrow$.*

proof. We consider saturation of the given TACE $\mathcal{A}$ and the goal clause $P_1(t_1), \ldots, P_n(t_n) \Rightarrow$ under basic ordered paramodulation wrt the ordering $\succ_{lpo}$ and the selection function sel_2 (defined in the proof of Proposition 17) and with eager ε-splitting. Following the same proof schema as for Proposition 17 (TAC) we show that, starting with $\mathcal{A} \cup P_1(t_1), \ldots, P_n(t_n) \Rightarrow$, every step of paramodulation returns either a clause smaller than all its premises (wrt to a well founded ordering $\gg$) or a clause of type (gf) or (df), where this latter clause type is similar to (gf) and also contains only a finite number of clauses (up to variable renaming).

The proof is nevertheless much more complicated than in the case of TAC (see Table 7 below). Indeed, like for TAC (Proposition 17), we obtain clauses of type (d_+) generalizing (d), in this case using basic narrowing. However, these clauses (d_+) can be combined, by resolution, with clauses of a type similar to (l) in Proposition 11. Clause decreasing, wrt $\gg$, is obtained for such resolution steps thanks to the restrictions on the equational theory considered.

Before we start the detailed proof of Proposition 22, 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

[3] For the sake of simplicity we denote $Q_1(x_1), x = u \Rightarrow Q(x)$ by $Q_1(x_1) \Rightarrow Q(u)$.

a lexicographic extension of orderings for each component. We denote by $|u|_z$ the number of occurrence of symbol z in term u. The following lemma will be used in the proof of saturation.

Lemma 23 *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: $X \cup Y$ where $X = \{x_1 = s_1, \ldots, x_n = s_n\}$ and $Y = \{y_1 = t_1, \ldots, y_m = t_m\}$ 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 $T = \{t_i = t_j \mid y_i = y_j, 1 \leq i, j \leq m\}$. 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 $X \cup Y \cup T$. Applying some replacements the initial system gets equivalent to (by abuse of notation σ is identified with the subsystem $\{z = z\sigma \mid z \in D\}$)

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

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 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 $mvar(t_{j+1}) < mvar(s)$. □

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:

- $m_1(C) = \infty$ if H is an equation, and is defined like in the proof of Proposition 17 otherwise,
- $m_2(C)$ is the number of equations in Ξ,
- $m_3(C)$ 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, It means in particular that this m_3 is unchanged by resolution, only left- and right-paramodulation on non variable terms may change m_3,
- $m_4(C)$ is the multiset of test predicate symbols occurring in Γ,
- $m_5(C) = m_5(s)$ if $H = Q(s)$, and $m_5(C) = (\{\}, 0)$ is the other cases,
- $m_6(C)$ is the multiset of the negative non-equational literals of $\Gamma\theta$.

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 $\mathbb{N}$.

Fact 24 below permits to conclude the proof of Proposition 22. 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, $Q_1, \ldots, Q_k, Q'_1, \ldots, Q'_m \in \mathcal{P}_0$.

Fact 24 *Starting from $\mathcal{A} \cup \{P_1(t_1), \ldots, P_n(t_n) \Rightarrow \}$, 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 24 is proved by a case analysis summarized in Figure 7 and detailed in Appendix D. In this analysis, we need to consider a new type of clauses (I_+) appearing during the saturation. Right paramodulation RP

with an equational clause $\Rightarrow f(\ell_1,\ldots,\ell_n) \to r$ into a clause of type (t) returns indeed a clause expanded into the following form [4]:

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

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 $\mathcal{P}_0$. □

Corollary 25 *GIP is decidable for TACE modulo a $\succ$-convergent sublinear and collapsing equational theory.*

7.4. *Example: Denning & Sacco's protocol*

Several security properties of the Denning & Sacco's protocol may be expressed as GIP wrt the TACE of Example 21: $Q_{01}(x) \Rightarrow$ expresses for instance that B has answered to a message not originating from A (authentication flaw) and $Q_{01}(S) \Rightarrow$ that the secret is revealed (confidentiality flaw). Both instances of GIP can be solved with the method of Proposition 22, revealing a known attack which involves only the agent B (and an attacker), and reveals the secret S.

Assume that the attacker initially knows A, $\mathsf{pub}(A)$, $\mathsf{pub}(B)$. This situation can be modeled adding some clauses of type (t) to $\mathcal{A}$, as explained in Example 21. With the attacker clauses of Example 21, the attacker is able to construct and send the message $m_0 = \mathsf{pair}(A, \mathsf{aenc}(A, \mathsf{pub}(B)))$ on the public channel Q, because this term is made of terms of its initial knowledge and public function symbols pair and aenc. It means that $m_0 \in L(\mathcal{A}, Q_{00})$. When the agent B reads m_0, he replies (in channel Q) with $\mathsf{enc}(S, \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(m_0), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(m_0))))$. Using the rewrite rules for projection (Example 8), 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 rewrite rules for asymmetric decryption, applied to the underlined 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 the channel Q.

The protocol can be patched in order to avoid such an attack, by requiring A to send the names A and B along with the symmetric key K $(j = 0, 1)$:

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

and by requiring the agent B to make some preliminary verifications on the message received before sending his answer, namely that the names match. This latter feature can be modeled in TAC by adding equations in the clause of type (d) representing B (with $i = 0, 1$ and $t = \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(x))))$:

$$Q_{i0}(x), \mathsf{snd}(\mathsf{fst}(t)) = x_B, \mathsf{fst}(\mathsf{fst}(t)) = \mathsf{fst}(x) \Rightarrow Q_{i1}(\mathsf{enc}(S, \mathsf{snd}(t)))$$

◇

7.5. *Example: recursive authentication protocol*

The *recursive authentication protocol* [25] ensures the distribution of certified session keys to a group of clients by a server which process recursively an unbounded list of requests. The automated verification of such group protocols has been studied in [4,22]. We shall follow below the presentation of [4], showing that it fits in our formalism. The server receives a sequence of requests for keys represented by a term of the form nil or [5]: $\langle \mathsf{hash}(\mathsf{m}(a), a, b, n_a, y), \langle a, b, n_a, y\rangle\rangle$, denoted below by $h_{\mathsf{m}_a}(a, b, n_a, y)$, where hash is a unary one-way function, a is the name of the principal requesting a certificate, b is the name of the principal with whom a is willing to share a key, n_a is a random number generated by a (nonce), $\mathsf{m}(a)$ is a mac key shared by the server and a and y is a subsequence of the other requests, which (if not nil) has the form $h_{\mathsf{m}_c}(c, a, n_c, y')$ (c

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

[5] We abbreviate $\mathsf{pair}(t_1, \mathsf{pair}(t_2, \ldots, \mathsf{pair}(t_{n-1}, t_n)))$ by $\langle t_1, \ldots, t_n\rangle$ $(n \geq 2)$.

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'	$\diagup$	d'/d_+	$=$	$>$	_	_	_	_						
RP	eq	t	l_+	$>$	_	_	_	_	_	$=$	$=$	$>$	_	_	_
LP	eq	d/d'	d'	$>$	_	_	_	_	_	$=$	$=$	$>$	_	_	_
LP	eq	g_+	g_+	$>$	_	_	_	_	_	$=$	$=$	$>$	_	_	_
R	_	t	$\diagup$												
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_+	_	$\diagup$												
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)

Fig. 7. Case analysis in the proof of Proposition 22.

is the name of another principal). The behaviour of the server, when receiving a request sequence, is defined by the following clauses of type (d) (where a, b, c, n_a, n_c are variables):

$$Q_0(x), x = h_{\mathsf{m}_a}(a, b, n_a, \mathsf{nil}) \Rightarrow Q_1\big(\mathsf{aenc}(\mathsf{pub}(a), \langle \mathsf{k}(a, b, n_a), b, n_a\rangle)\big)$$
$$Q_0(x), x = h_{\mathsf{m}_a}(a, b, n_a, h_{\mathsf{m}_c}(c, a, n_c, y')) \Rightarrow Q_1\big(\mathsf{aenc}(\mathsf{pub}(a), \langle \mathsf{k}(a, b, n_a), b, n_a\rangle)\big)$$
$$Q_1\big(\mathsf{aenc}(\mathsf{pub}(a), \langle \mathsf{k}(c, a, n_c), c, n_a\rangle)\big)$$

It means that the server sends to a one or two certificates encrypted with his public key, where k is a secret function used for the generation of session keys. Note the two occurrences of a in the equation of the second clause, which implicitly express an equality between the name of the requester of a query and the receiver in the next one. It is assumed that for the first element of the sequence, the receiver is actually the server himself (hence it is not necessary to send him a certificate). Moreover, we have a clause of type (t) for the enumeration of the requests by the server: $Q_0(x) \Rightarrow Q_0(\mathsf{next}(x))$, where next is an operator which pops the first element of a request's sequence, defined by the following collapsing equation (m is a variable): $\mathsf{next}(\mathsf{hash}(m, x_1, x_2, x_3, y), \langle x_1, x_2, x_3, y\rangle) = y$. ◇

8. Conclusion and further works

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

As future work we plan to extend the tree automata classes defined in this paper to disequality tests as in [9]. This would permit us to characterize languages of normal form wrt a TRS and is useful in particular in inductive theorem proving [1].

Equality tests between brother positions à la [8] can be easily incorporated into the Horn clauses representation of tree automata (see *e.g.* [12]). 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 [8] and [9] preserves emptiness decidability [26]. 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 also be interesting 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 us to axiomatize primitives like the exclusive-or.

Acknowledgments. We wish to thank Stéphanie Delaune for her contribution to early phases of this work, Anne-Cécile Caron, Sophie Tison Jean Goubault-Larrecq and Christopher Lynch for their feedback.

References

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

[2] T. Genet, F. Klay, Rewriting for Cryptographic Protocol Verification, in: Proc. of 17th Int. Conf. on Automated Deduction, CADE, Vol. 1831 of LNCS, Springer, 2000.

[3] F. Nielson, H. Riis Nielson, H. Seidl, Normalizable Horn Clauses, Strongly Recognizable Relations, and Spi, in: Static Analysis, 9th Int. Symp., SAS, Vol. 2477 of LNCS, Springer, 2002, pp. 20–35.

[4] R. Küsters, T. Wilke, Automata-Based Analysis of Recursive Cryptographic Protocols, in: 21st Annual Symp. on Theoretical Aspects of Computer Science, STACS, Vol. 2996 of LNCS, Springer, 2004, pp. 382–393.

[5] K. N. Verma, Two-Way Equational Tree Automata, Ph.D. thesis, ENS Cachan (Sep. 2003).

[6] H. Ohsaki, T. Takai, Decidability and Closure Properties of Equational Tree Languages, in: 13th Int. Conf. on Rewriting Techniques and Applications, RTA, Vol. 2378 of LNCS, Springer, 2002, pp. 114–128.

[7] H. Seki, T. Takai, Y. Fujinaka, Y. Kaji, Layered Transducing Term Rewriting System and Its Recognizability Preserving Property, in: Int. Conf. on Rewriting Techniques and Applications, RTA, Vol. 2378 of LNCS, Springer, 2002, pp. 98–113.

[8] B. Bogaert, S. Tison, Equality and Disequality Constraints on Direct Subterms in Tree Automata, in: 9th Symp. on Theoretical Aspects of Computer Science, STACS, Vol. 577 of LNCS, Springer, 1992, pp. 161–171.

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

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

[11] J. Goubault-Larrecq, Deciding $\mathcal{H}_1$ by Resolution, Information Processing Letters 95 (3) (2005) 401–408.
[12] F. Jacquemard, C. Meyer, C. Weidenbach, Unification in Extensions of Shallow Equational Theories, in: 9th Int. Conf. on Rewriting Techniques and Applications, RTA, Vol. 1379 of LNCS, Springer, 1998, pp. 76–90.
[13] N. Dershowitz, J.-P. Jouannaud, Rewrite systems, Elsevier, 1990, Ch. Handbook of Theoretical Computer Science, Volume B, pp. 243–320.
[14] L. Bachmair, H. Ganzinger, C. Lynch, W. Snyder, Basic Paramodulation, Information and Computation 121 (2) (1995) 172–192.
[15] H. Seidl, Haskell overloading is dexptime-complete, Information Processing Letters 52 (2) (1994) 57–60.
[16] F. Jacquemard, M. Rusinowitch, L. Vigneron, Tree Automata with Equality Constraints Modulo Equational Theories, in: U. Furbach, N. Shankar (Eds.), Proceedings of 3rd International Joint Conference on Automated Reasoning, IJCAR, Vol. 4130 of Lecture Notes in Artificial Intelligence, Springer, Seattle (WA), 2006, pp. 557–571.
[17] S. Tison, Tree automata and term rewrite systems, Invited tutorial at the 11th Int. Conf. on Rewriting Techniques and Applications, RTA (2000).
[18] R. Nieuwenhuis, A. Rubio, Paramodulation-Based Theorem Proving, Elsevier Science and MIT Press, 2001, Ch. Handbook of Automated Reasoning, Volume I, Chapter 7.
[19] A. Riazanov, A. Voronkov, Splitting Without Backtracking, in: Proc. of the 17th Int. Joint Conf. on Artificial Intelligence, IJCAI, Morgan Kaufmann, 2001, pp. 611–617.
[20] P. Devienne, J.-M. Talbot, S. Tison, Set-based analysis for logic programming and tree automata., in: P. V. Hentenryck (Ed.), Static Analysis, 4th International Symposium, SAS, Vol. 1302 of Lecture Notes in Computer Science, Springer, 1997, pp. 127–140.
[21] M. Abadi, C. Fournet, Mobile values, new names, and secure communication, in: 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2001, pp. 104–115.
[22] T. Truderung, Selecting theories and recursive protocols., in: M. Abadi, L. de Alfaro (Eds.), CONCUR, Vol. 3653 of Lecture Notes in Computer Science, Springer, 2005, pp. 217–232.
[23] C. Lynch, C. Meadows, On the relative soundness of the free algebra model for public key encryption., Electr. Notes Theor. Comput. Sci. 125 (1) (2005) 43–54.
[24] D. E. Denning, G. M. Sacco, Timestamps in Key Distribution Protocols, in: Communications of the ACM, 1981.
[25] J. A. Bull, D. J. Otway, The authentication protocol, Tech. rep., Defence Research Agency, Malvern, UK (1997).
[26] A.-C. Caron, H. Comon, J.-L. Coquidé, M. Dauchet, F. Jacquemard, Pumping, Cleaning and Symbolic Constraints Solving, in: 21st Int. Coll. on Automata, Languages and Programming, ICALP, Vol. 820 of LNCS, Springer, 1994, pp. 436–449.

Appendix A. Proof of Proposition 4 (TA)

We show that all the clauses generated by saturation of $\mathcal{A}$ and the goal clause $P_1(t_1), \ldots, P_n(t_n) \Rightarrow$ under ordered resolution wrt $\succ_{lpo}$ and the selection function sel_1 with eager application of the εsplit rule of Section 1 have the type (gs) (goal-subterm), or (gf), (goal-flat),

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.

Since the number of clauses of type (gs) and (gf) is exponential, the saturation terminates and GIP is solvable in deterministic exponential time.

Appendix B. Proof of Proposition 11 (TAE)

We detail below the different cases of saturation of $\mathcal{A} \cup \{P_1(t_1), \ldots, P_n(t_n) \Rightarrow\}$ under basic ordered paramodulation wrt the ordering $\succ_{lpo}$ and the selection function sel_1 and with eager ε-splitting.

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.

$\mathsf{LP}(\mathsf{eq},\mathsf{s})$ is not possible because the antecedents of clauses of type (s) contain only variables.

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

$\mathsf{R}(\mathsf{s},\mathsf{l})$ 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):

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

$\mathsf{R}(\mathsf{s},\mathsf{l})$ 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:

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

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

$\mathsf{R}(\mathsf{s},\mathsf{f})$ 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):

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

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

$\mathsf{R}(\mathsf{l},\_)$, $\mathsf{R}(\mathsf{f},\_)$: 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.

$\mathsf{R}(\mathsf{sp},\mathsf{l})$ returns a clause of type (l).

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

Altogether, all the clauses in saturation have type (l) or (f), and since there are only a finite number of clauses of these types, the saturation of $\mathcal{A} \cup \{P(t) \Rightarrow\}$ under basic ordered paramodulation terminates.

Appendix C. Proof of Proposition 17 (TAC) and Fact 18

We detail below the proof Fact 18 by a case analysis of all the instances of Eq, LP (R) and $\varepsilon\mathsf{split}$ involving as premises some clauses of type (t), (d), (d_+), (gf), (sp) or (g_+). We show that for each case, the conclusion of each such an instance is either of type (gf) (or (sp)) or either is smaller than all its premises wrt $\gg$.

$\mathsf{Eq}(\mathsf{d})$: 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 12 as the type of clauses obtained from clauses (d) and which contain no more equations.

$\mathsf{R}(\_, \mathsf{t})$: 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).

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

$\mathsf{R}(\_, \mathsf{d})$: 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 $m_1(\mathsf{sp}) = \infty$.

$\varepsilon\mathsf{split}(\mathsf{t})$ is not possible by definition.

$\varepsilon\mathsf{split}(\mathsf{d}_+)$: 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):

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

$\varepsilon\mathsf{split}(\mathsf{g}_+)$: 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 22 (TACE) and Fact 24

Let us detail in this Appendix the case analysis described in Figure 7, for the proof of Fact 24 and Proposition 22.

$\mathsf{Eq}(\mathsf{d})$: 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, $u_1, v_1, \ldots, u_k, v_k \in \mathcal{T}\big(\mathcal{F}, \{x_1, \ldots, x_n, x\}\big)$, 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.

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

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 17).

$\mathsf{R}(\mathsf{t},\mathsf{l}_+)$ 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, this latter clause must have the form $P_1(x),\ldots,P_m(x) \Rightarrow P(x)$ where x is a variable, otherwise, it would be split by ε-splitting. Moreover, P is distinct from $P_1,\ldots,P_m$, otherwise the clause would be a tautology.

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

$\mathsf{R}(\mathsf{t},\mathsf{d}_+)$ 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.

$\mathsf{R}(\mathsf{t},\mathsf{d}_+)$ is not possible when no negative literal is selected in the second premise (d_+), by definition of type (d_+) and because of the ordering strategy.

$\mathsf{R}(\mathsf{t},\mathsf{df})$ 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_+) not in (d_+) and 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 than for the premise (l_+).

Moreover, m_5 is strictly smaller for the conclusion than for the first premise (d_+). We have the following cases:

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 23 to s, ℓ_1 (which is linear by hypothesis) and r, and if follows that $m_5(s) > m_5(r\theta)$.

$\mathsf{R}(\mathsf{d}_+, \mathsf{d}_+)$ 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):

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

$\varepsilon\mathsf{split}(\mathsf{d}_+)$: 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):

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

$\varepsilon\mathsf{split}(\mathsf{g}_+)$: 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):

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