LIF

Laboratoire d'Informatique Fondamentale
de Marseille

Unité Mixte de Recherche 6166
CNRS - Université de Provence - Université de la Méditerranée

Multitrees Automata, Presburger's Constraints and Tree Logics

Denis Lugiez, Silvano Dal Zilio

Rapport/Report 08-2002

12 juin 2002

Multitrees Automata, Presburger's Constraints and Tree Logics

Denis Lugiez, Silvano Dal Zilio

Laboratoire d'Informatique Fondamentale

UMR 6166
CNRS - Université de Provence - Université de la Méditerranée

CMI 39 av. Joliot Curie, 13453 Marseille Cedex, France

lugiez,dalzilio@cmi.univ-mrs.fr

Abstract/Résumé

We describe *multitree automata* and a related logic on multitrees. Multitrees are extensions of trees with both associative and associative-commutative symbols that may bear arbitrary numbers of sons. An originality of our approach is that transitions of an automaton are restricted using Presburger's constraints. The benefit of this extension is that we generalize, all together, automata with equality and disequalities constraints as well as counting constraints. This new class of automata appears very general as it may encompass *hedge automata*, a simple yet effective model for XML schemata, *feature tree automata*, *automata with constraints between brothers* and *automata with arithmetical constraints*. Moreover, the class of recognizable languages enjoys all the typical good properties of traditional regular languages: closure under boolean operations and composition by associative and associative-commutative operators, determinisation, decidability of the test for emptiness, ...

We apply our automata to query languages for XML-like documents and to automated inductive theorem proving based on rewriting, obtaining each time new results. Using a classical connection between logic and automata, we design a decidable logic for (multi)trees that can be used as a foundation for querying XML-like document. This proposition has the same flavour as a query language for semi-structured recently proposed by Cardelli and Ghelli. The same tree logic is used to yield decidable cases of inductive reducibility modulo associativity-commutativity, a key property in inductive theorem proving based on rewriting.

Keywords: *tree automata, constraints, modal logic, query languages, automated theorem proving.*

Nous décrivons des automates pour multiarbres et une logique correspondante sur les multiarbres et ses applications. Les multiarbres sont des extensions des arbres avec des symboles associatifs-commutatifs et associatifs. Une originalité de l'approche est l'utilisation de contraintes de Presburger sur les transitions des automates. Ceci permet de généraliser les contraintes d'égalité et les contraintes arithmétiques. Cette classe généralise les *hedge automata* qui sont un modèle des schémas pour XML, les *feature-tree automata* et les *automates à contraintes entre frères*. Les langages reconnus possèdent toutes les bonnes propriétés des langages réguliers: cloture par les opérations ensemblistes et aussi par composition associative-commutative, et le vide est décidable.

Nous appliquons cette classes aux langages de requête pour XML et aux preuves par induction en démonstration automatique. En faisant le lien usuel entre logique et automates (pour des multiarbres) nous obtenons une logique décidable qui peut être utilisée comme langage de requête pour des documents XML. Cette logique est similaire à celle proposée par Cardelli et Ghelli. La même logique est utilisée pour donner des cas décidables de la propriété de réductibilité inductive modulo AC, une propriété clef pour la preuve de théorèmes inductifs utilisant la réécriture.

Relecteurs/Reviewers: R. Amadio

1 Introduction

The intimate relation between logic and automata has long been recognized, as attested for instance by Rabin's survey on decidable theories [Rab77]. The decidability of Presburger's arithmetic using word-automata is a striking example of this relation. Although word automata are very popular, tree automata are also successful for this purpose, even in the case of finite trees; see for example [Don70, TW68] for the connection between WSkS and regular tree languages. A chief reason for this success is that finite tree regular languages enjoy all the typical good properties of regular word languages (properties directly used for deciding the associated logic(s).) They can be combined using boolean connectives, they are determinisable, emptiness of recognized languages is efficiently decidable,...

Another important feature of (traditional) tree automata is their ability to be easily enriched with additional features, leading to better discriminative power. Nonetheless this flexibility is limited and, for example, non-linearity cannot be dealt with directly. A conventional example is that the set of instances of $f(t, t)$ is not a regular tree language. Also, regularity breaks down when useful axioms are considered, like closure of languages under associativity or associativity-commutativity of a function symbol. Finally, decision procedures derived from tree automata usually exhibit bad complexity. To circumvent these limitations, many variations around the tree automata principle have been designed, usually following the same basic ideas. For instance, [Ohs01] defines automata that use equations (mainly associativity-commutativity) to get the closure of accepted languages. Another classical extension is to consider equality constraints between subterms [BT92, CCD93, CCC$^+$94]. In some occasions, like with temporal logics [Var97], specific variations have also been designed in order to obtain decidability results with an optimal complexity.

Experience has proved that tree automata and related formalisms constitute an useful and versatile framework, which has been successfully applied to many different computer science areas. For example type inference and system analysis [AW92], evaluation of functional languages [Com00], automated theorem proving [CJ94], approximation of cryptographic protocols [CCM01], query processing for XML languages [Mur01], ... Nonetheless this approach to solving problems by extending the definition of tree automata has some drawbacks. Most particularly, extensions usually do not preserve some important properties, like closure under boolean operations or the decidability of the test for emptiness. Therefore it is important to build a formalism that is general enough while still preserving the good properties of regular tree automata.

In this paper, we define a new class of tree automata which combines closure under constraints, including equality constraints, and under associativity and associativity-commutativity axioms. The combination of both associativity-commutativity and equalities requires replacing trees by multitrees, which are roughly defined as multisets of trees where identical elements are grouped together. These automata can be considered as a generalization of *hedge automata* [PQ68], *feature tree automata* [NP93], *automata with constraint between brothers* [BT92] and *automata with arithmetical constraints* [LM94, Lug98] (the comparison is not exact since every classe works on slightly different structures.) An originality of our approach is that we take constraints to be regular expressions or Presburger's arithmetic formulas and that we consider a new kind of constraints, called Presburger's constraints. The constraints are described in a formalism reminding regular expression and contains equality and disequality tests, like $X = Y \oplus Y$ or $X \neq Y$, see Section 5.1. We prove a normal form property for our constraints language and the decidability of their first-order theory. Then, we revamp tree automata to fit the multitree structure combined with Presburger's constraints. Surprisingly enough we find that, after this simple adjustment, the class of recognizable languages is closed under the boolean operations and that the test for emptiness is decidable. The proof of this result heavily relies on the use (and the decidability) of Presburger's arithmetic. Actually, a similar phenomenon appears in the domain of unification modulo associativity-commutativity.

As a first application of multitree automata, we present a modal logic intended for querying semi-structured data [AB99], like XML documents, in which, by design, every formula directly relates to a multitree automaton. Our logic deliberately resembles (and extends on some points) *TQL*, a query language for semi-structured data based on the *ambient logic* [CG00, CG01]. In their presentation of *TQL*, the authors stick to a semantical point of view and consequently give only some axioms and computation rules for their language. In particular, they provide no results on the decidability of the satisfiability problem and give very few results on the computational aspects of their logic. We design a similar logic, *TL*, which adds new features like Presburger's constraints. In this extended framework it becomes possible to state that we try to match documents with *less than two fields for authors* or that contains *similar subfields*. Since the background of this logic is our new class of tree automata, we obtain immediately the decidability of model-checking, that is finding the answers to a query, and of the satisfiability problems, that is finding if a query is trivially empty. We can also infer results on *TQL* and on the (static fragment of) ambient logic. A similar technique could be applied to other modal logics with tree-like models, such as Pym's and O'Hearn's *bunch logic* [OP99]. This provides incidentally another example of the power of the (multitree) automata approach for logics in computer science.

To conclude, we put once again multitree automata to the test and use our approach to tackle inductionless induction methods in automated theorem proving [JK89]. In this framework, a key property is inductive reducibility, which is undecidable when associativity-commutativity is involved [KNRZ91]. Since we can express this property in *TL* for restricted non-linear cases, we get new decidable cases of this property. This result shed new light on the frontier between decidability and undecidability for this problem.

2 Terms and multitrees

Terms are constructed from a denumerable set X of variables and a finite set $\mathcal{F}$ of function symbols composed of a set F of free symbols, a set F_A of associative symbols and a set F_{AC} of associative-commutative symbols. For simplicity we shall assume that F_A contains only one symbol ., and that F_{AC} contains only one symbol $\oplus$ but all our results can be lifted to any number of distinct associative-commutative symbols. We use infix notation for . and $\oplus$.

Equality of terms denoted by $\equiv$, is defined modulo the associativity of . and associativity-commutativity of $\oplus$ as usual (see [DJ90] for instance). *Ground terms* are terms without variables. We shall work not on terms but on *multitrees*, which is a representation which takes into account the axioms: we consider . as the concatenation operation (on terms, not letters) and $\oplus$ as a multiset constructor. First we use the following rewrite rules to put terms in a standard form.

$$\begin{array}{lcl} (x.y).z & \to & x.(y.z) \\ (x \oplus y) \oplus z & \to & x \oplus (y \oplus z) \end{array}$$

Now we flatten a term $t_1 \oplus (t_2 \oplus \ldots \oplus t_n))$ by writing it as $t_1 \oplus \ldots \oplus t_n$ (i.e. $\oplus$ is seen as a variadic operator). Similarly we write $t_1.t_2 \ldots .t_n$ for $t_1.(t_2.\ldots.t_n))$.

The last step is to add equal terms appearing in a sum $t_1 \oplus t_2 \oplus \ldots \oplus t_n$ using commutativity and the rule $n.t \oplus m.s \to (n+m).t$ if $s \equiv t$. Initially $t_1 \oplus t_2 \oplus \ldots \oplus t_n$ is seen as $1.t_1 \oplus 1.t_2 \oplus \ldots \oplus 1.t_n$.

These rewrite processes terminate and yield normalized terms. The set $T_{\mathcal{F}}$ of ground normalized terms can be described by the grammar:

$$\begin{array}{lcll} T_{\mathcal{F}} & ::= & T_F \mid T_A \mid T_{AC} & \\ T_F & ::= & f(T_{\mathcal{F}}, \ldots, T_{\mathcal{F}}) & \text{with } arity(f) = n \geq 0 \\ T_A & ::= & (T_F \cup T_{AC}) \ldots . (T_F \cup T_{AC}) & \\ T_{AC} & ::= & n_1(T_F \cup T_A) \oplus \ldots \oplus n_p(T_F \cup T_A) & \end{array}$$

where we assume that sums involve only terms which are not equal.

Example 1 *From a term* $f((a.b).g(a), (a \oplus b) \oplus (b \oplus a))$, *we get first* $f(a.b.g(a), a \oplus b \oplus b \oplus a)$ *which is normalized into* $f(a.b.g(a), 2.a \oplus 2.b)$ *(for simplicity we write* x *and* a *instead of* $1.x$, $1.a$*). Notice that* $f(a.b.g(a), a \oplus 2.b \oplus a)$ *is not normalized.*

A *multitree* is a normalized term. Equality, still denoted by $\equiv$, is equality modulo permutation of arguments of $\oplus$, still denoted by $\equiv$, for instance $f(a.b.g(a), 2.a \oplus 2.b) \equiv f(a.b.g(a), 2.b \oplus 2.a)$. From now on, we shall consider only multitrees. In the remaining of the paper, we assume that the set of equivalence class of multitrees (for $\equiv$) of $T_F \cup T_A$ $T_F \cup T_A$ is enumerated as $e_1, e_2, \ldots$. Therefore each element of $T_{\mathcal{F}}$ is equal to a sum $\Sigma_i n_i e_i$ with the convention that $e_i = 1.e_i$ i.e. we consider the (infinite) sequence $e_1, e_2, \ldots$ as a set of generators and the natural numbers n_i as the coefficients (with respect to the generator set).

If $t = n_1 t_1 \oplus \ldots \oplus n_p t_p$ and $s = m_1 s_1 \oplus \ldots \oplus m_l s_l$, then, by definition, $s \oplus t$ denotes a multitree obtained by normalizing $m_1 s_1 \oplus \ldots \oplus m_l s_l \oplus n_1 t_1 \oplus \ldots \oplus n_p t_p$. For example $(2.a \oplus b) \oplus (b \oplus c)$ is $2.a \oplus 2.b \oplus c$.

Normalization is required to enable equality constraints: automata only have a finite memory, but since the $\oplus$ operator may have an arbitrary number of arguments, two identical arguments can be separated by an arbitrarily large distance exceeding the memory capacity of the automaton.

The *size* of $t = \Sigma_i n_i e_i$ is the number of non-zero n_i's, and the *module* of t is the maximal value of the n_i's. These notions induce a partial order on multitrees by comparing their sizes first and then their modules. The *minimal elements* of some set of multitrees are the elements minimal with respect to this order.

3 Presburger's arithmetic and semilinear sets

Let $\mathbb{N}$ be the set of natural numbers and let $+$ denote addition of natural numbers. Then the first-order theory of equality on this structure is called Presburger's arithmetic and is decidable. We shall often work with n-uples of integers (or terms) together with other objects and we use the vector notation (like $\vec{x}$) to make the distinction clearer.

Given $\vec{b} \in \mathbb{N}^n, P = \{\vec{p_1}, \ldots, \vec{p_m}\}$ with $\vec{p_i} \in \mathbb{N}^n$ the *linear set* $L(\vec{b}, P)$ is the set $\{\vec{x} \in \mathbb{N}^n \mid \vec{x} = \vec{b} + \Sigma_{i=1}^{i=m} \lambda_i \vec{p_i}, \lambda_i \in \mathbb{N}\}$. The element $\vec{b}$ is called the basis and the $\vec{p_i}$'s are the periods (if $P = \emptyset$, then $L(\vec{b}, P) = \{\vec{b}\}$). A *semilinear set* is a finite union of linear sets. Semilinear sets are closed under set operations and are the models of Presburger's arithmetic formulas.

3.1 Minimal solutions

The usual partial order on $\mathbb{N}^n$ is defined by $\vec{x} = (x_1, \ldots, , x_n) \leq \vec{y} = (y_1, \ldots, y_n)$ iff $\forall i, x_i \leq y_i$. The set of minimal elements of a set $E \subseteq \mathbb{N}^n$ is denoted by $Min(E)$[1]. This set is finite (By Dickson's lemma) and is equal to the solutions of $\vec{x} \in E \wedge \neg(\exists \vec{y} \in E \mid \vec{y} \leq \vec{x})$. Similarly, given a Presburger's arithmetic formula $\phi(x_1, \ldots, x_n)$, the set $Min\{(\alpha_1, \ldots, \alpha_n) \mid \phi(\alpha_1, \ldots, \alpha_n)\}$ is a effectively computable finite semilinear set. We say that $\vec{X}$ is *a solution* of $\varphi(\vec{X})$ if $\varphi(\vec{X})$ is true (which we denote by $\models \varphi(\vec{X})$).

Given some integer B, $Max_B \varphi(\vec{X})$ is used to bound B^{th} smallest solutions of φ. It is formally defined as follows:

- if φ has less than B solutions then $Max_B \varphi(\vec{X}) = Max\{x_i \mid \vec{X} = (x_1, \ldots, x_n) \text{ solution of } \varphi\}$
- Let $\vec{x_1}, \ldots, \vec{x_B}$ be minimal solutions of $\Psi(\vec{X_1}, \ldots, \vec{X_B})$ defined by

$$\bigwedge_{1 \leq i < j \leq B} X_i \neq X_j \wedge \bigwedge_{1 \leq i \leq B} \varphi(\vec{X_i})$$

  Then $Max_B(\varphi(\vec{X})) = Max_{1 \leq j \leq B}\{x_i^j \mid \vec{x}_j = (x_j^1, \ldots, x_j^n)\}$.

Since the value depends on the chosen minimal solutions, we can take $Max_B \varphi(\vec{X})$ to be the largest possible one. The main point is that $Max_B \varphi(\vec{X})$ is computable. This definition is used in sections 7.1, 7.4.

[1] beware that we have now two notions of minimality: one for terms and one for n-uples of natural numbers

3.2 Semilinear sets

We state now some properties on semilinear sets and Presburger's formula which are essential in the algorithm to decide emptiness of the language accepted by a multitree tree automaton.

Let L, M be a semilinear sets of $\mathbb{N}^n$, let $p \in \mathbb{N}$, then we define

- $L.M = \{\vec{z} = \vec{x} + \vec{y} \mid \vec{x} \in L, \vec{y} \in M\}$
- $L^p = \{\vec{z} = \vec{x}_1 + \ldots + \vec{x}_p \mid \vec{x}_i \in L\}$ and $L^0 = \{(0, \ldots, 0)\}$
- $L^+ = \bigcup_{i>0} L^i$ and $L^* = \bigcup_{i \geq 0} L^i$

Proposition 1 *$L.M$, L^p, L^+ and L^* are semilinear sets.*

The only non-trivial cases are the last ones. They are a consequence of the more general result on membership constraints stated in the next proposition.

Proposition 2 *The formula $\phi(\vec{x}, N)$ defined by $\vec{x} \in L^N$ with L a semilinear set of $\mathbb{N}^n$ is a effectively constructible Presburger's arithmetic formula.*

Proof Let $L = \bigcup_{i=1}^{i=l} L(\vec{b}_i, P_i)$ with $P_i = \{\vec{p}_{i,j} \mid j = 1, \ldots, l_i\}$. Then we claim that $\phi(\vec{x}, N)$ is equivalent to

$$\begin{array}{lll} \exists \mu_i, \lambda_{i,j} & \wedge(\vec{x} = \Sigma_{i=1}^{i=l}(\mu_i \vec{b}_i + \Sigma_{j=1}^{j=l_i} \lambda_{i,j} \vec{p}_{i,j})) & \text{/* } \vec{x} \text{ is a sum of sums of elements} \\ & & \text{of the linear sets composing } L\text{*/} \\ & \wedge \bigwedge_{i=1}^{i=l}((\bigvee_{j=1}^{j=l_i} \lambda_{i,j} \neq 0) \Rightarrow \mu_i \neq 0) & \text{/* one occurrence of period yields} \\ & & \text{an occurrence of the associated basis*/} \\ & \wedge(N = \Sigma_{i=1}^{i=l} \mu_i) & \text{/* the number of elements of linear sets} \\ & & \text{is the sum of occurrences of bases*/} \end{array}$$

- $\Rightarrow$ direction. Let $\vec{x} = \vec{x}_1 + \vec{x}_2 + \ldots + \vec{x}_N$ with $\vec{x}_j \in L(b_{i_j}, P_{i_j})$. Adding all $\vec{x}'_j s$ belonging to the same $L(\vec{b}_i, P_i)$, we get that $\vec{x} = \Sigma_{i=1}^{i=l}(\mu_i \vec{b}_i + \Sigma_{j=1}^{j=l_i} \lambda_{i,j} \vec{p}_{i,j})$ setting $\mu_i, \lambda_{i,j}$ to 0 if no $\vec{x}_j$ belongs to $L(\vec{b}_i, P_i)$, and with the additional facts that $N = \Sigma_{i=1}^{i=l} \mu_i$ and that if $\lambda_{i,j} \neq 0$ for some $j \in 1, \ldots, l_i$ then there is some $\vec{x}_j$ s.t. $\vec{x}_j \in L(\vec{b}_i, P_i)$ hence $\mu_i \geq 1$.
- $\Leftarrow$ direction. Let $\vec{x}, N$ satisfying the above formula. Then we can write $\vec{x} = \Sigma_{i \mid \mu_i \neq 0}(\vec{b}_i + \Sigma_{j=1}^{j=l_i} \lambda_{i,j} \vec{p}_{i,j}) + \underbrace{\vec{b}_i + \ldots + \vec{b}_i}_{\mu_i - 1 \ times}$, i.e $\vec{x} = \Sigma_{i | \mu_i \neq 0} \vec{x}_i$ with $\vec{x}_i \in L^{\mu_i}$, i.e. $\vec{x} \in L^N$.

□

Proposition 3 *The formula $(\vec{x} \in L_1^{N_1}.L_2^{N_2} \ldots L_p^{N_p}) \wedge \phi(\vec{x}, N_1, \ldots, N_p)$ with free variables $\vec{x}, N_1, \ldots, N_p$ where L_i's are semilinear sets of $\mathbb{N}^n$ and ϕ a Presburger's arithmetic formula, is a Presburger's arithmetic formula.*

Proof The formula is equivalent to $\exists \vec{x}_1 \ldots \vec{x}_p \ (\vec{x} = \vec{x}_1 + \ldots + \vec{x}_p) \wedge \vec{x}_1 \in L_1^{N_1} \wedge \ldots \wedge \vec{x}_p \in L_p^{N_p}) \wedge \phi(\vec{x}, N_1, \ldots, N_p)$ □

A immediate consequence is that the set of minimal values $(N_1, \ldots, N_p)$ such that $\exists \vec{x} \ \vec{x} \in L_1^{N_1}.L_2^{N_2} \ldots L_p^{N_p} \wedge \phi(\vec{x}, N_1, \ldots, N_p)$ is a finite computable set. Similarly, given some $N_1, \ldots, N_p$, the set of minimal values $(\vec{x}_1, \ldots, \vec{x}_p)$ such that $\vec{x}_1 \in L_1^{N_1} \wedge \vec{x}_2 \in L_2^{N_2} \ldots \wedge \vec{x}_p \in L_p^{N_p} \wedge \phi(\vec{x}_1, \ldots, \vec{x}_p, N_1, \ldots, N_p)$ is finite and computable. This property is used in the algorithm to decide the emptiness for tree automata.

4 Presburger's constraints

4.1 Definition

We define constraints on multitrees using a regular expression formalism.

Definition 1 *Given a set* $\mathcal{X} = \{X_1, \ldots,\}$ *of variables,* Presburger's constraints *are the formulae defined by:*

$$\phi ::= \vec{X} \in L_1^+ \ldots L_p^+ \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \exists X \ \phi$$

where $\vec{X} = (X_1, \ldots, X_n)$ *and* $L_1, \ldots, L_p$ *are semilinear sets of* $\mathbb{N}^n$*.*

The satisfiability relation $\models$ is defined by:

$$\vec{t} \models \vec{X} \in L_1^+ \ldots L_p^+ \text{ iff } \vec{t} = \Sigma_{i \in I_1} \vec{\lambda_i} e_i + \ldots + \Sigma_{i \in I_p} \vec{\lambda_i} e_i$$

where

$$\begin{cases} I_j \neq \emptyset \text{ for } j = 1, \ldots, p & \text{/*each } L_j \text{ occurs at least once*/} \\ i \in I_j \implies \vec{\lambda_i} \in L_j & \text{/*only elements of } L_j \text{ can occur*/} \\ \forall i, \vec{\lambda_i} \neq \vec{0} & \text{/* coefficients are non-trivial*/} \\ \forall j = 1, \ldots, n \ \exists \vec{\lambda}_{i_j} = (\lambda_1, \ldots, \lambda_j \neq 0, \ldots, \lambda_n) & \text{/*a component } j \text{ can't be } \Sigma_k 0.e_k\text{*/} \end{cases}$$

The satisfiability relation for non-atomic or quantified formulas is defined as usual. We shall write $\models \phi(\vec{t})$ when $\vec{t}$ satisfies $\phi(\vec{X})$.

4.2 Expressivity

A possible extension of this class of constraints is to allow L^* instead of L^+ with the obvious semantics (the set of indices i such that $\vec{\lambda_i} \in L$ can be empty). This doesn't change the expressivity of formulas since $\vec{X} \in L_1^+ \ldots L_p^+ L^*$ is equivalent to $\vec{X} \in L_1^+ \ldots L_p^+ L^+ \vee \vec{X} \in L_1^+ \ldots L_p^+$.

Presburger constraints are rich enough to express systems of equations and disequations in multitrees. Given an equation $\Sigma_i \alpha_i X_i = \Sigma_j \beta_j X_j$ on free variables $X_1, \ldots, X_n$, with $\alpha_i, \beta_j \in \mathbb{N}$, let L be the semilinear set corresponding to the solutions of this equation in $\mathbb{N}^n$. Then the solution of the equation (interpreted in multitrees) is the set of n-uples of multitrees satisfying $\vec{X} \in L^+$. The closure of Presburger's constraints under conjunction and negation yields the claim.

In Presburger's arithmetic[2], we can't express $x > y$ with a quantifier-free formula if $>$ is not a primitive predicate. But it can be easily defined by $(x, y) \in L$ for L a semilinear set model of $\exists z \ x = y + z + 1$. The same situation occurs here since we can define $X > Y$ for multitrees by $\exists Z \ X = Y + Z$ which corresponds to $(X, Y) \in L^+ M^*$ for M a semilinear set model of $\exists z \ x = y + z$ (L as previously).

4.3 Normal forms for quantifier free formula

Presburger's constraints enjoy a kind of normal form property.

Proposition 4 *Each quantifier free Presburger's constraint is equivalent to a disjunction of atomic formulas* $\vec{X} \in L_1^+ \ldots L_p^+$*.*

Proof The proof is by structural induction on the formulas.

[2] in this paper, we take Presburger's arithmetic with non-negative numbers only

1. Negation of atomic formulas. We claim that $\neg(\vec{X} \in L_1^+ \dots L_p^+)$ is equivalent to

$$\begin{array}{ll} \vee & \vec{X} \in (\overline{L_1 \cup \dots \cup L_p})^+ (L_1 \cup \dots \cup L_p)^* \\ \vee & \bigvee_{i=1}^{i=p} (\vec{X} \in ((L_1 \cap \overline{L_i})^* \dots (L_{i-1} \cap \overline{L_i})^* (L_{i+1} \cap \overline{L_i})^* \dots (L_p \cap \overline{L_i})^*)) \end{array}$$

   (where $\overline{L}$ denotes the complementation of L).

   Let $\vec{t}$ such that $\vec{t} \not\models L_1^+ \dots L_p^+$. By definition, $\vec{t} = \Sigma_j \vec{\lambda}_j e_j$ and

   - Either there is one $\vec{\lambda}_j$ belonging to none of the L_i's (other $\vec{\lambda}_j$ may also be in the same case or not). This is handled by the first member of the disjunction.
   - Or each $\vec{\lambda}_j$'s are in some L_k, but there is at least one L_i such that no $\vec{\lambda}_i$ belong to L_i. This is handled by the i^{th}disjunct of the second member.

2. Conjunction of atomic formula. $\vec{X} \in L_1^+ \dots L_p^+ \wedge \vec{X} \in M_1^+ \dots M_q^+$ is equivalent to

$$\bigvee_{\substack{\{i_1, \dots, i_l\} = \{1, \dots, p\} \\ \{j_1, \dots, j_l\} = \{1, \dots, q\}}} \vec{X} \in (L_{i_1} \cap M_{j_1})^+ \dots (L_{i_l} \cap M_{j_l})^+$$

   (several i_k's can be identical, and similarly for j_k's but each L_i and each M_j occurs at least once).

3. General case. Given $\phi = \neg\phi_1$ or $\phi = \phi_1 \wedge \phi_2$, by induction hypothesis we can assume that ϕ_i $(i = 1, 2)$ are disjunction of atomic formulas. Then the same reasoning as above yields the result.

□

4.4 Satisfiability of Presburger's constraints

The normal form property allows to state:

Proposition 5 *The satisfiability of quantifier free Presburger's constraints is decidable.*

Proof By proposition 4, we can restrict ourself to atomic constraints.
Let $\vec{X} \in L_1^+ \dots L_p^+$ be an atomic constraint. We can assume that the L_i's are linear sets. Otherwise if $L = M_1 \cup \dots \cup M_p$ with linear sets M_i's, the constraint is equivalent to the disjunction of the atomic constraints obtained by replacing L by $M_1^* \dots M_{i-1}^* M_i^+ M_{i+1}^* \dots M_p^*$. The constraint is unsatisfiable iff there is a component i in $1, \dots, n$ such that all $\vec{\lambda}_j$'s are zero on this component. Since $L_j = L(\vec{b}_j, \{\vec{p}_1^j, \dots, \vec{p}_{m_j}^j\})$, this amounts to saying that the i^{th} components of b_j and of the p_l^j's for $j = 1, \dots, p$, $l = 1, \dots, m_j$, are zero. In this case any n-tuple $\vec{t}$ satisfying the constraint is such that $t_i = \Sigma_i 0.e_i$.

Conversely, if this is not the case, there exists $L_j = L(\vec{b}_j, \{\vec{p}_1^j, \dots, \vec{p}_{m_j}^j\})$ such that the i^{th} component of $\vec{b}_j$ or of some $\vec{p}_l^j$ is non-zero. Then there exists $\vec{t}$ such that $\models \varphi(\vec{t})$ and $t_i \neq 0$ (choose some $\vec{\lambda} = \vec{b}_j + \vec{p}_l^j$) which states that the formula is satisfiable. □

We show now that this result actually holds for any first-order Presburger's constraint. This is obtained by showing that a formula $\exists X_i \vec{X} \in L_1^+ \dots L_p^+$ is equivalent to a disjunction of atomic constraints. The idea is to realize that this quantifier elimination process can be done by simply erasing the i^{th} component in the L_i as this is done for the decidability of Presburger arithmetic using the automata approach. However this procedure may be incorrect when the expression $L_1^+ \dots L_p^+$ allows n-uples which have 0 on their i^{th} components.

Example 2 *Let* $L = L(\begin{pmatrix} 0 \\ 1 \end{pmatrix}, \{\begin{pmatrix} 1 \\ 1 \end{pmatrix}\})$

Let us consider the formula $\exists X_1 \ (X_1, X_2) \in L$. *Eliminating* X_1 *and the first component gives* $X_2 \in L/1 = L((1), \{(1)\})$. *Then* $X_2 = 1$ *satisfies this latter constraint but there is no pair* (X_1, X_2) *with* $X_2 = 1$ *satisfying the first one: to avoid* $X_1 = 0$ *requires that we use a strictly positive multiple of the period. The problem disappears if we transform the constraint into the equivalent one* $\exists X_1 \ (X_1, X_2) \in L'$ *with* $L' = L(\begin{pmatrix} 1 \\ 2 \end{pmatrix}, \{\begin{pmatrix} 1 \\ 1 \end{pmatrix}\})$. *We have added the period to the basis to get a new basis which doesn't raise any problem.*

To overcome this problem, we show that each atomic formula $\varphi(\vec{X})$ is equivalent to a disjunction of atomic formulas where it is ensured that the i^{th} component can't be 0 (for some given i).

Given a semilinear set L, we have $L = (L \cap (X_i = 0)) \cup (L \cap X_i > 0)$ where $X_i = 0$ (resp. $X_i > 0$) denotes the semilinear set of n-uples with i^{th} component equal to 0 (resp. $X_i > 0$). The intersection with L is semilinear and we can replace L^+ by $(L \cap (X_i > 0))^*(L \cap (X_i = 0))^+$ or $(L \cap (X_i = 0))^*(L \cap (X_i > 0))^+$. Iterating this process combined with the elimination of terms L^*, we can replace an atomic constraint by a disjunction of $\vec{X} \in L_1^+ \dots L_p^+$ where the i^{th} component of each element is 0 or the i^{th} component of each element is greater than 0[3]. We assume that at least one L_j has non-zero i^{th} component (otherwise the constraint is unsatisfiable). Let let L/i denote $\{(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) \mid \exists x_i \ (x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n) \in L\}$ and $\vec{X}/i$ denote $(X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_n)$. The following proposition allows to eliminate existential quantifiers.

Proposition 6 *Let* L_j*'s be as above, then* $\exists X_i \ \vec{X} \in L_1^+ \dots L_p^+$ *is equivalent to* $\vec{X}/i \in L_1/i^+ \dots L_p/i^+$.

From which we get:

Proposition 7 *The satisfiability of Presburger's constraints is decidable.*

Proof The proof is a simple structural induction on formulas. □

This result can also be obtained by encoding the constraints into skolem arithmetic which is the theory of natural numbers with multiplication and is decidable (private communication from Achim Blumensath).

5 Multitree automata with Presburger's constraints

We describe our class of *multitree automata* with Presburger's constraint and prove some basic properties: closure under basic operations, determinisation, and decidability of the test for emptiness of the recognized language. Recall that the set of function symbol over which we operate is $\mathcal{F} = F \uplus F_A \uplus F_{AC}$ and that, for the sake of brevity, we restrict our study to a unique associative symbol, $\cdot$, and a unique AC symbol, $\oplus$.

5.1 Definition and Transition Relation

A multitree automaton with Presburger's constraint is a triple $\mathcal{A} \triangleq \langle Q, Q_{fin}, R\rangle$ where Q is a finite set of states and R is a set of transition relations. We suppose that Q is the disjoint union of four simpler sets, Q_A, Q_{AC}, Q_F and Q_{N2AC}. In this setting, Q_{AC} is associated to the set F_{AC} of AC symbols (similarly for A and F), while Q_{N2AC} is a set of auxiliary states used for transitions from A, F to AC symbols. We also suppose that the set Q_{fin} of final states does not contain auxiliary states, that is $Q_{fin} \subseteq Q_F \uplus Q_A \uplus Q_{AC}$. Finally, we decompose the set R into four distinct categories, (type 1) to (type 4) below.

(type 1) $\phi(X_1, \dots, X_n) : \ f(q_1, \dots, q_n) \to q$ with $q_i \in Q, \ q \in Q_F$
$\phi(X_1, \dots, X_n)$ a Presburger's constraint

[3] this property can be established with a direct reasoning on the basis and the periods

(type 2) $\mathcal{L}(\mathbf{A}_q) \rightarrow q$ with $q \in Q_A$
$\mathbf{A}_q$ is a finite state word automaton on the alphabet $Q_{AC} \cup Q_F$
$L(\mathbf{A}_q)$ denotes the word language accepted by $\mathbf{A}_q$

(type 3) $\psi(N): \; Nq \rightarrow q'$ with $q \in Q_F \cup Q_A, q' \in Q_{N2AC}$,
ψ a Presburger's arithmetic formula.

(type 4) $\psi(N_1, \ldots, N_p): \; N_1 q_1 \oplus \ldots \oplus N_p q_p \rightarrow q$ with $Q_{N2AC} = \{q_1, \ldots, q_p\}$
ψ a Presburger's arithmetic formula.

Before defining the transition relation we give some intuitions on the different types of rules. Type 1 transition rules are similar to those found in traditional tree automata, with the notable addition of the Presburger constraint $\phi(X_1, \ldots, X_n)$. This constraint allows expresssing very liberal inequalities, disequalities and counting constraints between brothers. For example, the rule $(X_1 = X_2) : f(q_1, q_2) \rightarrow q$ may fire only if we are at a node labeled f with equivalent left and right children. Type 2 rules are almost orthogonal to the rest of the presentation and could be added to other variations on tree automata without many complications. These rules use regular word languages over the states of the automaton, as in $(q_2 + q_3).q_1^* \rightarrow q$ for example. This idea is directly inspired by *hedge automata* [Mur01] and it has been added with the (successful) intention to directly encompass hedge automaton in our framework. Type 3 and 4 rules are extension of type 1 rules for associative and AC function symbols. Type 3 rules allow us to express arithmetical constraints on the number of associative and free symbols. They "produce" auxiliary states (in Q_{N2AC}.) Type 4 rules are the equivalent for AC symbols and "consume" auxiliary states. The side condition $Q_{N2AC} = \{q_1, \ldots, q_p\}$ ensures that all auxiliary states are taken into account. More formally, the transition relation $\rightarrow_\mathcal{A}$ is the smallest relation such that:

$$
\begin{array}{rcl}
t = f(t_1, \ldots, t_n) \rightarrow_\mathcal{A} q & \text{if} & t_i \rightarrow_\mathcal{A} q_i \text{ for } i = 1, \ldots, n \\
 & & \models \phi(t_1, \ldots, t_n) \\
 & & \phi(X_1, \ldots, X_n): \; f(q_1, \ldots, q_n) \rightarrow q \in R \\
t = t_1.t_2. \; \ldots \; .t_m \rightarrow_\mathcal{A} q & \text{if} & t_i \rightarrow_\mathcal{A} q_i \in Q_F \cup Q_{AC} \text{ for } i = 1, \ldots, m \\
 & & q_1.q_2. \; \ldots \; .q_m \in \mathcal{L}(\mathbf{A}_q) \\
 & & \mathcal{L}(\mathbf{A}_q) \rightarrow q \in R \\
t = nt' \rightarrow_\mathcal{A} q & \text{if} & t' \rightarrow_\mathcal{A} q' \\
 & & \models \psi(n) \\
 & & \psi(N): \; Nq' \rightarrow q \in R \\
t = \Sigma_{i=1}^{i=p}(\Sigma_{j=1}^{N_i} n_{i,j} t_{i,j}) \rightarrow_\mathcal{A} q & \text{if} & n_{i,j} t_{i,j} \rightarrow_\mathcal{A} q_i \text{ for } j = 1, \ldots, N_i \\
 & & \models \psi(N_1, \ldots, N_p) \\
 & & \psi(N_1, \ldots, N_p): \; N_1 q_1 \oplus \ldots \oplus N_p q_p \rightarrow q \in R
\end{array}
$$

A multitree is accepted if $t \rightarrow_\mathcal{A} q$ with $q \in Q_{final}$, and $\mathcal{L}(\mathcal{A})$ the language accepted by $\mathcal{A}$ is the set of accepted multitrees. A straightforward consequence of the definition is that $t \equiv t'$ implies that $t \rightarrow_\mathcal{A} q$ iff $t' \rightarrow_\mathcal{A} q$. Note that we can keep the transition relation of $\mathcal{A}$ when we add a new state q_{p+1} in $Q_F \cup Q_A$ (like some forthcoming constructions do) by adding a new term $N_{p+1} q_{p+1}$ and a new constraint $N_{p+1} = 0$ in type 4 rules.

Example 3 *In this example, we use regular expressions and equations or disequations for constraints, which are equivalent to but more explicit than word automata or membership constraints. Given* $F = \{a, b, pair(\_, \_)\}$*, we define* $\mathcal{A}$ *by* $Q_F = \{q_e, q_o\}, Q_A = \{q_A\}, Q_{N2AC} = \{q'_e, q'_o\}, Q_{AC} = \{q_S\}$*, with* $Q_{Final} = \{q_S\}$*, and the rules:*

$a \to q_e$ *and* $b \to q_e$
$a \to q_o$ *and* $b \to q_o$
$(Q_F \cup Q_{AC})^* \to q_A$
$X_1 = X_2 : pair(\_,\_) \to q_e$ /* *elements of pair must be equal to reach* q_e */
$X_1 \neq X_2 : pair(\_,\_) \to q_o$ /* *elements of pair must be distinctsto reach* q_o */
$N = 1 : Nq_e \to q'_e$ /* *only one copy of an element of* q_e */
$\exists N'\ N = 2N' : Nq_o \to q'_o$ /* *only an even number of copies of elements of* q_o */
$N_e = 1 \wedge N_o \geq 1 : N_e q'_e + N_o q'_o \to q_S$ /* *only one* q'_e*, any number of* q'_o */
(where $\_$ *stands for any state in the relevant set).*

Then the multitrees $pair(a.b, a.b) \oplus 2pair(a, a.b)$ *and* $pair(a \oplus 2b, a \oplus 2b) \oplus 2pair(a, a.b)$ *are accepted but not* $2pair(a, a) \oplus pair(a.b, b)$*.*

A simpler definition of multitree automata is obtained by replacing type 3 and type 4 rules by rules:

$$\psi(N_1, \dots, N_p) :\ N_1 q_1 \oplus \dots \oplus N_p q_p \to q$$

where $t = \Sigma_k n_k t_k \to q$ using the above rule if $\models \psi(m_1, \dots, m_p)$ where $m_i = \Sigma_{t_k \to q_i} n_k$. Unfortunately this natural definition forbids the determinization of automata, and it is not even clear whether the intersection of two recognizable languages is recognizable. This means that the corresponding class of automata probably doesn't enjoy good properties.

An apparently richer formalism is to allow type 4 rules

$$\phi(X) \wedge \psi(N_1, \dots, N_p) :\ N_1 q_1 \oplus \dots \oplus N_p q_p \to q$$

where ϕ is a Presburger's constraint. Actually these rules can be simulated in the previous setting. Given a constraint $\phi(X)$ of the form $\vec{X} \in L_1^+ \dots L_m^+$ we construct a rule

$$\psi'(N_1, \dots, N_p) :\ N_1 q_1 \oplus \dots \oplus N_p q_p \to q$$

equivalent to

$$\phi(X) \wedge \psi(N_1, \dots, N_p) :\ N_1 q_1 \oplus \dots \oplus N_p q_p \to q$$

Firstly, for each state q_i, for each type 3 rule $\phi(N) : Nq'_i \to q_i$ we introduce m states q_i^j $(j = 1, \dots, m)$ and the rules $N \in L_j \wedge \phi(N) : Nq'_i \to q_i^j$. Finally we add the rule:

$$\bigwedge_{i=1}^{i=p} N_i = \Sigma_j N_i^j \wedge \bigwedge_{j=1}^{j=m} \Sigma_{i=1}^{i=p} N_i^j \geq 1 \wedge \psi(N_1, \dots, N_p) : N_1^1 q_1^1 \oplus \dots \oplus N_p^m q_p^m \to q$$

6 Basic operations on constrained automata

6.1 Completion

A automaton is *complete* iff each multitree reaches some state. From an automaton $\mathcal{A}$ we can build a complete automaton $\mathcal{B}$. First we add four sink states $q_s \in Q_F, q_{s,A} \in Q_A, q_{s,N2AC} \in Q_{N2AC}, q_{s,AC} \in Q_{AC}$, then we add the following new rules:

type 1 $True :\ f(q_1, \dots, q_n) \to q_s$ for each sequence $q_1, \dots, q_n$ with $q_i \in Q_F \cup Q_A \cup Q_{AC} \cup \{q_s, q_{s,A} q_{s,AC}\}$,

type 2 $L(\mathbf{A}_{q_{s,A}}) \to q_{s,A}$ for $a_{q_{s,A}}$ an automaton accepting $(Q_F \cup Q_{AC})^*$.

type 3 $True :\ Nq_s \to q_{s,N2AC}$, $True :\ Nq_{s,A} \to q_{s,N2AC}$,

type 4 $N_s > 0 :\ N_1 q_1 \oplus \dots \oplus N_p q_p \oplus N_s q_{s,N2AC} \to q_{s,AC}$

and we replace the rules

$$\psi(N_1,\ldots,N_p):\ N_1q_1 \oplus \ldots \oplus N_pq_p \to q \text{ of } \mathcal{A}$$

by rules

$$\psi(N_1,\ldots,N_p) \wedge N_s = 0 \wedge N_{s,N2AC} = 0:\ N_1q_1 \oplus \ldots \oplus N_pq_p \oplus N_sq_{s,N2AC} \to q$$

Proposition 8 *The automaton $\mathcal{B}$ is complete and $\mathcal{L}(B) = \mathcal{L}(\mathcal{A})$.*

6.2 Closure under union

Given two automata $\mathcal{A}$ and $\mathcal{A}'$, a automaton $\mathcal{B}$ accepting $\mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{A}')$ is obtained by taking the union of the automata. A slight change required is to replace (type 4) rules of $\mathcal{A}$

$$\psi(N_1,\ldots,N_p):\ N_1q_1 \oplus \ldots \oplus N_pq_p \to q$$

by

$$\psi(N_1,\ldots,N_p) \wedge \bigwedge_j N'_j = 0:\ N_1q_1 \oplus \ldots \oplus N_pq_p \oplus N'_1q'_1 \oplus \ldots \oplus N_mq'_m \to q$$

(and similarly for $\mathcal{A}'$)

Proposition 9 *The automaton $\mathcal{B}$ accepts $\mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{A}')$.*

6.3 Closure under intersection

The classical product construction is used to get an automaton $\mathcal{B}$ accepting $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$.

- The set of states is $Q_F \times Q'_F \cup Q_A \times Q'_A \cup Q_{N2AC} \times Q'_{N2AC} \cup Q_{AC} \times Q'_{AC}$.
- The set of final states is $Q_F \times Q'_F$.
- The set of rules $R_{\mathcal{B}}$ is as follows.

(type 1) $(\phi \wedge \phi')(X_1,\ldots,X_n):\ f((q_1,q'_1),\ldots,(q_n,q'_n)) \to (q,q') \in R_{\mathcal{B}}$

if $\phi(X_1,\ldots,X_n):\ f(q_1,\ldots,q_n) \to q \in R_{\mathcal{A}}$
$\phi'(X_1,\ldots,X_n):\ f(q'_1,\ldots,q'_n) \to q' \in R_{\mathcal{A}'}$

(type 2) $L(\mathbf{A}_q \times \mathbf{A}_{q'}) \to (q,q') \in R_{\mathcal{B}}$

where $\mathbf{A}_q \times \mathbf{A}_{q'}$ is the product automaton accepting $\{(q_i,q'_j) \mid q_i \in L(\mathbf{A}_q), q'_j \in L(\mathbf{A}_{q'})$.

(type 3) $(\psi \wedge \psi')(N) : N(q_1,q_2) \to (q'_1,q'_2) \in R_{\mathcal{B}}$

if $\psi(N) : Nq_1 \to q'_1 \in R_{\mathcal{A}}$,
$\psi'(N) : Nq_2 \to q'_2 \in R_{\mathcal{A}'}$

(type 4) $\psi(\Sigma_jN_{1j},\ldots,\Sigma_jN_{pj}) \wedge \psi'(\Sigma_iN_{i1},\ldots,\Sigma_iN_{ip'})\ : N_{1,1}(q_1,q'_1) \oplus \ldots \oplus N_{p,p'}(q_p,q'_{p'}) \to (q,q') \in R_{\mathcal{B}}$

if $\psi(N_1,\ldots,N_p):\ N_1q_1 \oplus \ldots \oplus N_pq_p \to q \in R_{\mathcal{A}}$,
$\psi'(N_1,\ldots,N_{p'}):\ N_1q'_1 \oplus \ldots \oplus N_{p'}q'_p \to q' \in R_{\mathcal{A}}$

Proposition 10 *The automaton $\mathcal{B}$ accepts $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$.*

Proof We show that $t \rightarrow_{\mathcal{B}} (q,q')$ iff $t \rightarrow_{\mathcal{A}} q$ and $t \rightarrow_{\mathcal{A}'} q'$. The proof is by structural induction on multitrees and is similar to the proof for tree automata. Only the case of type 4 rule is slightly more complex. Assume $t = \Sigma_i t_i \rightarrow (q,q')$ using

$$\psi(\Sigma_j N_{1j},\ldots,\Sigma_j N_{pj}) \wedge \psi'(\Sigma_i N_{i1},\ldots,\Sigma_i N_{ip'}) : N_{11}(q_1,q_1') \oplus \ldots \oplus N_{pp'}(q_p,q_{p'}') \rightarrow (q,q')$$

Then $t = \Sigma_{i=1}^{i=n_{1,1}} t_{1,1}^i \oplus \ldots \oplus \Sigma_{i=1}^{i=n_{p,p'}} t_{p,p'}$ with $t_{1,1}^i \rightarrow (q_1,q_1')$ for $i = 1,\ldots,n_{1,1},\ldots,t_{p,p'}^i \rightarrow (q_p,q_{p'}')$ for $i = 1,\ldots,n_{p,p'}$ and $\models \psi(\Sigma_j n_{1j},\ldots,\Sigma_j n_{pj})$. Therefore $t \rightarrow_{\mathcal{A}} q$ (and similarly for q' and $\mathcal{A}'$).

Conversely assume that $t \rightarrow_{\mathcal{A}} q$ using $\psi(N_1,\ldots,N_p):\ N_1q_1 \oplus \ldots \oplus N_pq_p \rightarrow q$ and $t \rightarrow_{\mathcal{A}'} q'$ using $\psi'(N_1,\ldots,N_p):\ N_1q_1' \oplus \ldots \oplus N_pq_p' \rightarrow q'$.

We have $t = t_1 \oplus \ldots \oplus t_n$ with $\begin{cases} n_1 \textit{ multitrees } t_k's \textit{ reach } q_1 \\ \ldots \\ n_p \textit{ multitrees } t_k \textit{ reach } q_p \\ \models \psi(n_1,\ldots,n_p) \end{cases}$ *and* $\begin{cases} n_1' \textit{ multitrees } t_k's \textit{ reach } q_1' \\ \ldots \\ n_p' \textit{ multitrees } t_k's \textit{ reach } q_p' \\ \models \psi'(n_1',\ldots,n_p') \end{cases}$

We write t as $\Sigma_{i,j}\Sigma_{\substack{t_k \rightarrow_{\mathcal{A}} q_i \\ t_k \rightarrow_{\mathcal{A}'} q_j'}} t_k$ and we define $m_{i,j}$ as the number of t_k appearing in the subsum. By definition we have $n_i = \Sigma_j m_{i,j}$ and $n_j' = \Sigma_i m_{i,j}$. Therefore $t \rightarrow_{\mathcal{B}} (q,q')$. □

6.4 Determinization

An automaton $\mathcal{A}$ is *deterministic* if for each t there is at most one state q such that $t \rightarrow_{\mathcal{A}} q$. We describe how to compute an equivalent deterministic tree automaton with a variant of the classical subset construction. The basic idea is similar to the idea for determinization of automata with constraints between brothers [BT92], but the Presburger's arithmetic constraints require some additional work: if M distincts multitrees s_i (of the form $n.t$) reach the state $Q = \{q_1,\ldots,q_l\}$ in the deterministic automaton, meaning that each s_i reaches all the q_j's of Q in the non-deterministic automaton, we must find out which states q the multitree $s_1 \oplus \ldots \oplus s_M$ can reach in the non-deterministic automaton. Therefore we decompose $s_1 \oplus \ldots \oplus s_M$ into all possible expressions

$$\underbrace{\Sigma_{s_i\ reach\ q_1} s_i}_{x_1\ element} \oplus \ldots \oplus \underbrace{\Sigma_{s_i\ reach\ q_l} s_i}_{x_l\ element}$$

(some x_j can be zero, but $M = \Sigma_{j=1,\ldots,l} x_j$) and check which states these expressions can reach.

6.4.1 Determinization of condition

We recall how to get rid of the non-determinism arising from the overlapping of conditions of rules. Let $\phi_1,\ldots,\phi_n$ be formula on free variables $X_1,\ldots,X_m$, then for each $I \subseteq \{1,...,n\}$, we define $\phi_I = \bigwedge_{i\in I}\phi_i \wedge \bigwedge_{j\notin I}\neg\phi_j$. Then $\phi_I \wedge \phi_J$ is unsatisfiable if $I \neq J$ and ϕ_i is equivalent to $\bigvee_{I\ni i}\phi_I$. Computing the ϕ_I's and replacing a rule with condition ϕ by the rules with condition ϕ_I will be called determinization of conditions. Clearly, this operation doesn't modify the language accepted by the automaton.

6.4.2 The subset construction

The deterministic automaton $\mathcal{A}_{Det}$ is constructed from the non-deterministic complete automaton $\mathcal{A}$ as follows.

- The set of states Q_{Det} is the union of $Q_F^{Det} = 2^{Q_F}$, $Q_A^{Det} = 2^{Q_A}$, $Q_{N2AC}^{Det} = 2^{Q_{N2AC}}$ and $Q_{AC}^{Det} = 2^{Q_{AC}}$.
- The final states are the states $\mathcal{Q} \in Q_{Det}$ containing a final state of $\mathcal{A}$.
- The set of rules $R_{\mathcal{A}_{Det}}$ is obtained as follows.

1. Perform the determinization of conditions $\phi(X_1, \ldots, X_n) : f(q_1, \ldots, q_n) \to q$ of type 1 rules. For simplicity we still denote by $R_{\mathcal{A}}$ the resulting set of rules. Then

$$\phi_I(X_1, \ldots, X_n) : f(\mathcal{Q}_1, \ldots, \mathcal{Q}_n) \to \mathcal{Q} \in R_{\mathcal{A}_{Det}}$$

if $\mathcal{Q} = \{q \in Q_F \mid \exists q_1 \in \mathcal{Q}_1, \ldots, q_n \in \mathcal{Q}_n \; s.t. \; \phi_I(X_1, \ldots, X_n) : f(q_1, \ldots, q_n) \to q \;\in R_{\mathcal{A}}\}$.

2. For each $\mathcal{Q}_A \in Q_A$, for each $q \in \mathcal{Q}_A$,
   - Transform the automaton $\mathbf{A}_q$ with alphabet $Q_F \cup Q_{AC}$ into an automaton $\mathbf{A}_q^R$ on the alphabet $Q_F^{Det} \cup Q_{AC}^{Det}$ as follows [4]: $\mathbf{A}_q^R$ has the same set of states as $\mathbf{A}_q$, has the same initial state, the same set of final states, it contains the transition $s.\mathcal{Q} \to s'$ iff there exists a transition $s.q \to s'$ in $\mathbf{A}_q$ with $q \in \mathcal{Q}$.
   - For each $\mathcal{Q} \subseteq \mathcal{Q}_A$, compute $\mathbf{A}_{\mathcal{Q}}^{Det}$ a (not necessarily deterministic) automaton accepting $\bigcap_{q \in \mathcal{Q}} L(\mathbf{A}_q^R) \cap \bigcap_{q \notin \mathcal{Q}} \overline{L(\mathbf{A}_q^R)}$.

   Then

$$L(\mathbf{A}_{\mathcal{Q}}^{Det}) \to \mathcal{Q} \in R_{\mathcal{A}_{Det}}$$

(type 3) Perform the determinization of condition on type 3 rules $\psi(N) : Nq \to q' \in R_{\mathcal{A}}$. For simplicity we still denote by $R_{\mathcal{A}}$ the resulting set of rules. Then

$$\psi_I(N) : N\mathcal{Q} \to \mathcal{Q}' \in R_{\mathcal{A}_{Det}}$$

if $\mathcal{Q}' = \{q' \mid \exists q \in \mathcal{Q} \; s.t. \; \psi_I(N) : Nq \to q' \in R_{\mathcal{A}}\}$

(type 4) Let $Q_{N2AC}^{Det} = \{\mathcal{Q}_1, \ldots, \mathcal{Q}_l\}$, and let $Ind(\mathcal{Q}_i) = \{k \mid q_k \in \mathcal{Q}_i\}$. To each $\psi(N_1, \ldots, N_p)$ condition of type 4 rules, we associate $\psi'(M_1, \ldots, M_l)$ defined as

$$\exists x_i^j \bigwedge_{i=1}^{i=l} (M_i = \Sigma_{j \in Ind(Q_i)} x_j^i) \wedge \psi(\Sigma_{i \;\mid\; 1 \in Ind(Q_i)} x_1^i, \ldots, \Sigma_{i \;\mid\; p \in Ind(Q_i)} x_p^i)$$

and we compute the formula ψ_I by determinizing the ψ_i' $(i = 1, \ldots, n)$. Then

$$\psi_I(M_1, \ldots, M_l) : M_1\mathcal{Q}_1 \oplus \ldots \oplus M_l\mathcal{Q}_l \to \mathcal{Q} \in R_{\mathcal{A}_{Det}}$$

if $\mathcal{Q} = \{q \mid \exists i \in I \; s.t. \; \psi_i(N_1, \ldots, N_p) : N_1 q_1 \oplus \ldots \oplus N_p q_p \to q \in R_{\mathcal{A}}\}$

Proposition 11 *The automaton $\mathcal{A}_{Det}$ is deterministic and $\mathcal{L}(\mathcal{A}_{Det}) = \mathcal{L}(\mathcal{A})$.*

Proof We use $\to_{\mathcal{A}}$ to denote the transition relation of $\mathcal{A}$ and $\to_{\mathcal{A}_{Det}}$ to denote the transition relation of $\mathcal{A}_{Det}$. We show by structural induction on t that $t \to_{\mathcal{A}_{Det}} \mathcal{Q}$ iff $\mathcal{Q} = \{q \mid t \to_{\mathcal{A}} q\}$.

1. Let $t = f(t_1, \ldots, t_n)$ for $(n \geq 0)$. By induction hypothesis, for each $j = 1, \ldots, n$ we have $t_j \to_{\mathcal{A}_{Det}} \mathcal{Q}_J$ iff $\mathcal{Q}_j = \{q_k \mid t_j \to_{\mathcal{A}} q_k\}$.

   (a) Let $\psi_I(t_1, \ldots, t_n)$ be the unique condition such that $\models \psi_I(t_1, \ldots, t_n)$ (only conditions of relevant rules are considered). If $f(t_1, \ldots, t_n) \to_{\mathcal{A}} q$, there is a rule $\psi_I(X_1, \ldots, X_n) : f(q_1, \ldots, q_n) \to q \;\in R_{\mathcal{A}}$ such that $t_i \to_{\mathcal{A}} q_i$ and $\models \psi_I(t_1, \ldots, t_n)$. If $f(t_1, \ldots, t_n) \not\to_{\mathcal{A}} q'$, then for all rule $\psi_K(X_1, \ldots, X_n) : f(q_1', \ldots, q_n') \to q'$ either there is some i such that $t_i \not\to_{\mathcal{A}} q_i'$ (i.e. $q_i' \notin Q_i$ or $\not\models \psi_K(t_1, \ldots, t_n)$. This implies that $t'_{\mathcal{A}_{Det}} Q$ using $\psi_I(X_1, \ldots, X_n) : f(\mathcal{Q}_1, \ldots, \mathcal{Q}_n) \to \mathcal{Q}$ with $Q = \{q \mid t \to_{\mathcal{A}} q\}$.

[4] R for refined, since the new alphabet can be seen as a refinement of the old one

(b) Conversely, assume that $t \rightarrow_{\mathcal{A}_{Det}} \mathcal{Q}$ and let $\psi_I(X_1,\ldots,X_n) : f(\mathcal{Q}_1,\ldots,\mathcal{Q}_n) \rightarrow \mathcal{Q}$ be the last applied rule. By definition $t_i \rightarrow_{\mathcal{A}_{Det}} \mathcal{Q}_i$ and $\models \psi_I(t_1,\ldots,t_n)$. Therefore $t \rightarrow_{\mathcal{A}} q$ for each rule $\psi_I(X_1,\ldots,X_n) : f(q_1,\ldots,q_n) \rightarrow q$ and $t \not\rightarrow_{\mathcal{A}} q'$ for any other rule $\psi_K(X_1,\ldots,X_n) : f(q'_1,\ldots,q'_n) \rightarrow q'$ such that either $q_i \notin Q_i$ or $I \neq K$.

Therefore $t \rightarrow_{\mathcal{A}_{Det}} \mathcal{Q}$ iff $\mathcal{Q} = \{q \mid t \rightarrow_{\mathcal{A}} q\}$.

2. First, we show the correctness of the construction of the automata for the refined alphabet. Let q_A be in Q_A, let s_0 denote the initial state of $\mathbf{A}_{q_A}$, $\rightarrow_{\mathbf{A}_{q_A}}$ (resp. $\rightarrow_{\mathbf{A}^R_{q_A}}$) denote the transition relation of $\mathbf{A}_{q_A}$ (resp. $\mathbf{A}^R_{q_A}$). We show that

$$\{s \mid s_0.\mathcal{Q}_1\ldots\mathcal{Q}_m \rightarrow_{\mathbf{A}^R_{q_A}} s\} = \{s \mid s_0.q_1\ldots q_m \rightarrow_{\mathbf{A}_{q_A}} s \textit{ for } q_1 \in \mathcal{Q}_1,\ldots,q_m \in \mathcal{Q}_m\}$$

by induction on m.

- $m = 1$. $s_0.\mathcal{Q}_1 \rightarrow_{\mathbf{A}^{Det}_{q_A}} s'$ implies $s_0.q \rightarrow_{\mathbf{A}_{q_A}} s'$ for some $q \in \mathcal{Q}$. Conversely if $s_0.q \rightarrow_{\mathbf{A}_q} s'$ for $q \in \mathcal{Q}_1$ then $s_0.\mathcal{Q}_1 \rightarrow_{\mathbf{A}_q} s'$.
- Assume that the property holds up to $m-1$. By definition, $\{s \mid \mathcal{Q}_1\ldots\mathcal{Q}_m \rightarrow_{\mathbf{A}^R_{q_A}} s\} = \{s' \mid s_0.\mathcal{Q}_1\ldots\mathcal{Q}_{m-1} \rightarrow_{\mathbf{A}^R_{q_A}} s'' \textit{ and } s''.\mathcal{Q}_m \rightarrow \mathbf{A}^R_{q_A} s\}$. Let s'' be such that $s_0.\mathcal{Q}_1\ldots\mathcal{Q}_{m-1} \rightarrow_{\mathbf{A}^R_{q_A}} s''$, then $s''.\mathcal{Q}_m \rightarrow_{\mathbf{A}^{Det}_{q_A}} s'$ implies that $q \in \mathcal{Q}_m$, $s''.q \rightarrow s'$. Conversely if $s''.q \rightarrow_{\mathbf{A}_{q_A}} s'$ with $q \in Q_m$ then $s_0.\mathcal{Q}_1\ldots\mathcal{Q}_m \rightarrow_{\mathbf{A}^R_{q_A}} s'$.

Let us show now the property on multitrees and states. Let $t = t_1.\ \ldots\ .t_m$ and assume that $t_i \rightarrow_{\mathcal{A}_{Det}} \mathcal{Q}_i$. By definition $t \rightarrow_{\mathcal{A}_{Det}} \mathcal{Q}_A$ iff $Q_1.\ldots.Q_m \rightarrow_{\mathbf{A}^R_{q_A}} s$ for s a final state of $\mathbf{A}^R_{q_A}$, $q_A \in \mathcal{Q}_A$, and $Q_1.\ldots.Q_m \not\rightarrow_{\mathbf{A}^R_{q_A}} s$ for s a final state of $\mathbf{A}^R_{q_A}$ if $q_A \notin \mathcal{Q}_A$.

Since a word $Q_1.\ \ldots\ .Q_m$ reaches a unique Q_A, the same property holds for t .

3. Let $t = n.s$.

By induction hypothesis, $s \rightarrow_{\mathcal{A}_{Det}} \mathcal{Q}$ iff $Q = \{q \mid s \rightarrow_{\mathcal{A}} q\}$.

(a) By definition $t \rightarrow_{\mathcal{A}} q'$ iff there is a rule $\psi_I(N) : Nq \rightarrow q' \in R_{\mathcal{A}}$ such that $\models \psi_I(n)$ and $s \rightarrow_{\mathcal{A}} q$. If $\mathcal{Q}' = \{q' \mid t \rightarrow_{\mathcal{A}} q'\}$ then we have $t \rightarrow_{\mathcal{A}_{Det}} \mathcal{Q}'$.

(b) Conversely, if $t \rightarrow_{\mathcal{A}_{Det}} \mathcal{Q}'$ using $\psi_I(N) : N\mathcal{Q} \rightarrow \mathcal{Q}'$ then $s \rightarrow_{\mathcal{A}} q$ iff $q \in \mathcal{Q}$ and $\models \psi_I(n)$. Therefore $t \rightarrow_{\mathcal{A}} q'$ for all $q' \in \mathcal{Q}'$. There is no other rule $\psi_I(N) : Nq \rightarrow q'$ such that $t \rightarrow_{\mathcal{A}} q'$ using this rule as the last one since either $q \notin Q$ or $\not\models \psi_I(\lambda)$.

Therefore $t \rightarrow_{\mathcal{A}_{Det}} \mathcal{Q}'$ iff $\mathcal{Q}' = \{q \mid t \rightarrow_{\mathcal{A}} q\}$.

4. Let $t = \Sigma_k s_k$ where each s_k has the form $n_k t_k$. By induction hypothesis, $s_k \rightarrow_{\mathcal{A}_{Det}} Q_i$ iff $Q_i = \{q_i \mid s_k \rightarrow_{\mathcal{A}} q_i\}$.

(a) We can write $t = \Sigma_{i=1}^{i=l}\Sigma_{s_k \rightarrow_{\mathcal{A}_{Det}} Q_i} s_k$. Let $M_i = |\{s_k \mid s_k \rightarrow_{\mathcal{A}_{Det}} \mathcal{Q}_i\}|$. We have $t \rightarrow_{\mathcal{A}} q$ iff there is a type 4 rule $\psi(N_1,\ldots,N_p) : N_1 q_1 \oplus \ldots \oplus N_p q_p \rightarrow q \in R_{\mathcal{A}}$ such that

$$\begin{cases} \textit{for any } i \in 1,\ldots,l, j \in \textit{Ind}(\mathcal{Q}_i), \exists x^i_j \textit{ s.t. } M_i = \Sigma_{j \in Ind(\mathcal{Q}_i)} x^i_j, \\ \models \phi(\Sigma_{1 \in Ind(\mathcal{Q}_i)} x^i_1,\ldots,\Sigma_{p \in Ind(\mathcal{Q}_i)} x^i_p) \end{cases}$$

Therefore $t \rightarrow_{\mathcal{A}_{Det}} \mathcal{Q}$ with $\mathcal{Q}$ the set of states q of rules satisfying the previous conditions.

(b) Conversely assume that $t \rightarrow_{\mathcal{A}_{Det}} \mathcal{Q}$, with $\psi_I(M_1,\ldots,M_l) : M_1 Q_1 \oplus \ldots \oplus M_l Q_l \rightarrow Q$ the last rule applied. Then $t = \Sigma_{i=1}^{i=l}\Sigma_{s_k \rightarrow_{\mathcal{A}_{Det}} Q_i} s_k$ where $|\{s_k \rightarrow_{\mathcal{A}_{Det}} Q_i\}| = M_i$. Moreover for $i = 1,\ldots,l$ there exist x^i_j's such that $M_i = \Sigma_{j \in Ind(\mathcal{Q}_i)} x^i_j$ and $\models \phi(\Sigma_{1 \in Ind(\mathcal{Q}_i)} x^i_1,\ldots,\Sigma_{p \in Ind(\mathcal{Q}_i)} x^i_p)$ with $\phi(N_1,\ldots,N_p) : N_1 q_1 \oplus \ldots \oplus N_p q_p \rightarrow q$ a type 4 rule. Combined with the induction hypothesis, this implies that $t \rightarrow_{\mathcal{A}} q$ for all $q \in \mathcal{Q}$ and that $t \not\rightarrow_{\mathcal{A}} q$ if $q \notin Q$.

$\square$

6.4.3 Closure under complementation

Let $\mathcal{A}$ be a complete deterministic automaton (by previous results, we can always compute such an automaton accepting the same language as $\mathcal{A}$). A automaton $\mathcal{B}$ accepting the complement of $\mathcal{L}(\mathcal{A})$ is obtained from $\mathcal{A}$ by exchanging final and non final states.

6.5 Closure under concatenation and sum

The closure under concatenation is straightforward and directly follows from the closure of regular word languages under concatenation.

The closure under the $\oplus$ operation is more complex. Given $\mathcal{A}$ and $\mathcal{A}'$ (supposed to be complete) we construct an automaton $\mathcal{B}$ such that $\mathcal{L}(\mathcal{B}) = \{t \mid t \equiv t_1 \oplus t_2, t_1 \in \mathcal{L}(\mathcal{A}), t_2 \in \mathcal{L}(\mathcal{A}')\}$. A slight difficulty arises in this case: for instance if $t = 2a \oplus b \in \mathcal{L}(\mathcal{A})$ and $t' = a \oplus c \in \mathcal{L}(\mathcal{A}')$, the multitree corresponding to $t \oplus t'$ is $3a \oplus b \oplus c$. Therefore $\mathcal{B}$ must be able to split $3a$ into $2a$ and a (at type 3 rule level) and to transfer this information at type 4 rule level. For simplicity we assume that the multitrees accepted by $\mathcal{A}$ and $\mathcal{A}'$ are all in T_{AC} (when it is not the case, a construction similar to the next one can be done).

First $\mathcal{B}$ contains the same states and rules as the automaton for the intersection except that the set of final states is empty. Now we add a new state $(q, q')^{\oplus}$ for each pair (q, q') and the rule

$$\exists N_1 > 0, N_2 > 0 : N = N_1 + N_2 \wedge \phi(N_1) \wedge \phi'(N_2) : N(q, q') \to (q_{N2AC}, q'_{N2AC})^{\oplus}$$

if $\phi(N) : Nq \to q_{N2AC}$ is a rule of $\mathcal{A}$ and $\phi'(N) : Nq' \to q'_{N2AC}$ is a rule of $\mathcal{A}'$.

Then we add a state $q_{Success}$ which is final and the rules:

$$\begin{array}{rll} \exists N^1_{(q,q')}, N^2_{(q,q')} : & \wedge & \bigwedge_{(q,q')} N_{(q,q')} = N^1_{(q,q')} + N^2_{(q,q')} \\ & \wedge & \phi(\Sigma_{q'} N^1_{(q_1,q')} + \Sigma_{q'} N^{\oplus}_{(q_1,q')}, \ldots, \Sigma_{q'} N^1_{(q_p,q')} + \Sigma_{q'} N^{\oplus}_{(q_p,q')}) \\ & \wedge & \phi'(\Sigma_q N^2_{(q,q'_1)} + \Sigma_q N^{\oplus}_{(q,q'_1)}, \ldots, \Sigma_q N^1_{(q,q'_{p'})} + \Sigma_q N^{\oplus}_{(q,q'_{p'})}) : \end{array}$$

$$N_{(q_1,q'_1)}(q_1, q'_1) \oplus \ldots \oplus N_{(q_p,q'_{p'})}(q_p, q'_{p'}) \oplus N_{(q_1,q'_{1'})^{\oplus}}(q_1, q'_{1'})^{\oplus} \oplus \ldots \oplus N_{(q_p,q'_{p'})^{\oplus}}(q_p, q'_{p'})^{\oplus} \to q_{Success}$$

if $\phi(N_1, \ldots, N_p) : N_1 q_1 \oplus \ldots \oplus N_p q_p \to q \in R_{\mathcal{A}}$ with q a final state (and similarly for $\mathcal{A}'$).

Proposition 12 $\mathcal{L}(\mathcal{B}) = \{t \mid t \equiv t_1 \oplus t_2, t_1 \in \mathcal{L}(\mathcal{A}), t_2 \in \mathcal{L}(\mathcal{A}')\}$

Proof Assume that $t = n_1 t_1 \oplus \ldots \oplus n_l t_l \to q$ in $\mathcal{A}$ with $\phi(N_1, \ldots, N_p) : N_1 q_1 \oplus \ldots \oplus N_p q_p \to q$. Moreover we assume that the last rule used for $n_i t_i$ in this derivation is $\phi_i(N) : N q'_{j_i} \to q_{j_i}$. A similar situation occurs for $s = m_1 s_1 \oplus \ldots \oplus m_{l'} s_{l'} \to q$ and $\mathcal{A}'$.

$$t \oplus s = \Sigma_{s_i \equiv t_i} (n_i + m_i) t_i \oplus \Sigma_{\in I} n_i t_i \oplus \Sigma_{j \in J} m_j s_j$$

$n_i + m_i$ satisfies $\exists N_1 > 0, N_2 > 0 : N = N_1 + N_2 \wedge \phi(N_1) \wedge \phi'(N_2)$ then $(n_i + m_i) t_i \to (q_{k_i}, q'_{k'_i})^{\oplus}$.

For t_i occurring in the second sum, we have $t_i \to (q_{k_i}, q')$ (the value of q'_k is irrelevant). The number of terms reaching $(q_i, \_)$ in the first and second sum, is the number of terms reaching q_i in t. Similarly for the first and third sum and s and $\mathcal{A}'$. Therefore we can use the new type 4 rule to get $t \to q_{Success}$.

Conversely, assume that $t \to q_{Success}$ in $\mathcal{B}$. We can write

$$t = u_1 \oplus \ldots \oplus u_{m_1} \oplus v_1 \oplus \ldots \oplus v_{m_2} \oplus w_1 \oplus \ldots \oplus w_{m_3}$$

where $u_i \to (q_{j_i}, q'_{j'_i})^{\oplus}$ for $i = 1, \ldots, m_1$, $v_i \to (q_{j_i}, \_)$ for $i = 1, \ldots, m_2$, $w_i \to (\_, q_{j_i})$ for $i = 1, \ldots, m_1$ and the for each q_i the number of u's and v's reaching a state $(q_i, \_)^{\oplus}$ or $(q_i, \_)$ is what is needed to satisfy $\phi(N_1, \ldots, N_p)$ (similarly for u's, w's, $\mathcal{A}'$ and $\phi'(N_1, \ldots, N_p)$.

Moreover each $u_i \to (q_{j_i}, q'_{j_i})^{\oplus}$ means that $u_i = (n_i + m_i) t_i$ such that $n_i t_i \to q_{j_i}$ in $\mathcal{A}$ and $m_i t_i \to q'_{j'_i}$. Therefore we can split t into s accepted by $\mathcal{A}$ and s' accepted by $\mathcal{A}'$.

$\square$

7 Bounds for multitree automata

This section gives technical results which are required by the algorithm which decides the emptiness of the language accepted by a multitree automaton.

The basic idea of the algorithm is classical: from a set of marked states we compute all the reachable states and mark them until we reach a fixed point. But the problem is more complex here. Because of the constraints, we may have to require that more than one multitree reach a state before marking it. For instance in automata with constraints between brothers (which are a particular subcase of our class), given a rule $X_1 \neq X_2 : f(q, q) \rightarrow q'$, to have one multitree reaching q is not enough to mark q' (since the constraint can't be satisfied if the set of multitrees reaching q consists of one unique term). The state q' will be marked only when more than two multitrees reach q (two multitrees are not enough because it doesn't ensure the conservation of the invariant, but three multitrees is since it ensure not only that q' is reachable but also that it is reached by at least three multitrees). Therefore the algorithm computes a static bound which indicates how many multitrees must reach a state before marking it. Unfortunately, another complication arises in our case because multitrees like $\Sigma_{k=1}^{k=p} n_i t_i$ can be arbitrarily large since $\oplus$ can have any number of arguments. This is why we compute a bound which is a pair: the first element of the pair bounds the *size of multitrees* (i.e. p), the other one bounds the *module of multitrees* i.e. $Max_{i=1,\ldots,p}(n_i.)$.

To set up these bounds, we discuss each type of rule from the easiest case to the hardest one. From now on, we assume that $\mathcal{A}$ is a deterministic complete automaton. In the following, the determinism ensures that if a multitree reaches a state q it can't reach another state q'.

7.1 Type 2 rules

Type 2 rules are easy to handle. Let $t_1. \ \ldots \ .t_m \rightarrow q_A$ with the rule $L(\mathbf{A}_{q_A}) \rightarrow q_A$. Assume that $t_i \rightarrow q_i$ for $i = 1, \ldots, m$ and that there is one q_i, say q_{i_0} which is reached by more than B multitrees. Then we construct more than B distinct terms reaching q_A (replace t_{i_0} by any term reaching q_{i_0}).

7.2 Type 3 rules

Let $\psi(N) : \ Nq' \rightarrow q$ be a type 3 rule R where we assume that $\psi(N)$ is satisfable. If more than B multitrees reach q', then there are more than B multitrees reaching q. Moreover, given some B, we can compute $M_R(B) = Max_B(\psi(N)$ (see definition in section 3.1).

7.3 Type 4 rules

Let $\psi(N_1, \ldots, N_p) : N_1 q_1 \oplus \ldots \oplus N_p q_p \rightarrow q$ be a type 3 rule R. Let B_R be a the maximal value of the components of minimal solutions of $\psi(N_1, \ldots, N_p)$. Assume that $t = t_1 \oplus \ldots \oplus t_m$ reaches q using R as the last rule, and thet t is minimal in size. We say that t and t' are independent if we don't have $t = n.s''$ and $t' = m.s''$. Let q_i be the (unique) state reached by t_i. If there are at least $B_R + 1$ independent multitrees reaching q_i, then there are at least $C_{B_R+1}^{B_R}$ distinct multitrees reaching q: in t there are at most B_R such multitrees, therefore we have at least $C_{B_R+1}^{B_R}$ possible choices to construct distinct multitrees reaching q with R like s.

Example 4 *Let R be $N_1 + N_2 \geq 2 : N_1 q_1 \oplus N_2 q_2 \rightarrow q$. The minimal solutions of ψ are $(2, 0), (1, 1), (0, 2)$ hence $B_R = 2$. Assume that we have computed three ($3 = B_R + 1$) independent multitrees s_1, s_2, s_3 that reach q_1, and that we have found that $s_1 \oplus s_2$ reach q. Then we know that there are $C_3^2 = 3$ distinct terms reaching q in the same way: $s_1 \oplus s_2, s_2 \oplus s_3, s_1 \oplus s_3$*

7.4 Type 1 rules

Let R be a type 1 rule

$$\vec{X} \in L_1^+ \ldots L_m^+ : f(q_1^R, \ldots, q_n^R) \rightarrow q_F^R$$

The normalization property (proposition 4) ensures that we can consider only such conditions. For simplicity, we shall assume that each q_i is some q_l^R in Q_{AC}. Actually the case of $q_i \in Q_F \cup Q_A$ is simpler, one simply replace corresponding sums in the proof by sums consisting of a single element. We merge condition of type 4 rules to have a unique rule

$$\Psi_j(N_1, \ldots, N_p) : N_1 q_\oplus \ldots \oplus : N_p q_p \to q_j^{AC}$$

for each q_j^{AC}. Similarly, we assume that for each pair q', q' there is a unique type 3 rule:

$$\psi_{q',q}(N) : Nq' \to q$$

Given $t = f(t_1, \ldots, t_n)$ reaching q using R as the last rule. We have $\vec{t} = \Sigma_k \vec{\lambda}_k e_k$ and we show now that the $\vec{\lambda}_k$'s must belong to some particular specific semilinear sets and that they are finitely many such sets.

Let $q \in Q_F \cup \mathcal{Q}_A$, let $\vec{q} = (q_1, \ldots, q_n)$ be a n-uple of elements of Q_{N2AC}. For simplicity we denote by $\psi_l(N) : Nq \to q_l$ the type 3 rule associated to the pair q, q_l. We define $L_{i,q,\vec{q}} \subseteq \mathbb{N}^n$ as:

$$L_{i,q,\vec{q}} = \{\vec{x} = (x_1, \ldots, x_n) \mid \left\{ \begin{array}{l} \vec{x} \in L_i \\ \models \psi_l(x_l) \ for\ l = 1, \ldots, n \end{array} \right. \}$$

The last technical notation is needed to distinguish between zero and non-zero components in a n-uple $\vec{\lambda}$ (because $\Sigma_k 0.e_k$ is not a term). Given $J \subseteq \{1, \ldots, n\}$ we define the semilinear set $L_{i,q,\vec{q},J} \subseteq \mathbb{N}^n$ as:

$$L_{i,q,\vec{q},J} = \{\vec{x} = (x_1, \ldots, x_n) \mid \left\{ \begin{array}{l} \vec{x} \in L_{i,q,\vec{q}} \\ \forall j \in J\ x_j = 0\ and\ \forall j \notin J\ x_j > 0 \end{array} \right. \}$$

The multitree t reaches q_F^R with R the last rule applied iff

$$\vec{t} = \Sigma_{q \in Q_A \cup Q_F} \Sigma_{e_k \to q} \vec{\lambda}_k e_k$$

with

$$\begin{array}{l} \forall k, \exists i\ \vec{\lambda}_k \in L_i\ and\ \forall i = 1, \ldots, m,\ \exists k\ \vec{\lambda}_k \in L_i \\ \forall l = 1, \ldots, n\ t_l = \Sigma_{q \in Q_A \cup Q_F} \Sigma_{e_k \to q} \lambda_k^l e_k \to q_l^R \end{array}$$

where $\vec{\lambda}_k = (\lambda_k^1, \ldots, \lambda_k^n)$.

Let

$$\vec{\Lambda}_{i,q,\vec{q},J} = \Sigma_{\begin{array}{l} e_k \to q \\ \vec{\lambda}_k \in L_{i,q,\vec{q},J} \end{array}} \vec{\lambda}_k = (\Lambda^1_{i,q,\vec{q},J}, \ldots, \Lambda^n_{i,q,\vec{q},J}) \in L_{i,q,\vec{q},J}^{\alpha_{i,q,\vec{q},J}}$$

(fix q, then sum all $\vec{\lambda}_k$'s in the same $L_{i,q,\vec{q},J}$'s for all possible choices of $i, \vec{q}, J$).

Similarly for $i = 1, \ldots, n$, $k = 1, \ldots, |Q_{N2AC}|$ let α_l^k be the number of distinct (non-zero) subterms of t_l reaching $q_k \in Q_F \cup Q_A$, and let $\vec{\alpha}_l = (\alpha_l^1, \ldots, \alpha_l^{|Q_{N2AC}|})$. By definition of $L_{i,q,\vec{q},J}$ we have

$$: \alpha_l^k = \Sigma_{q \in Q_A \cup Q_F} \Sigma_{\begin{array}{l} i = 1, \ldots, m \\ l \notin J \\ l^{th}\ component\ of\ \vec{q}\ is\ q_k \end{array}} \alpha_{i,q,\vec{q},J}$$

Let us denote $\Psi_l(N_1, \ldots, N_p) : N_1 q_1 \oplus \ldots \oplus q_p \to q_l^R$ the last rule used in $t_l \to q_l^R$. By the above remarks, an equivalent formulation of the conditions on $\vec{t}$ is:

$$\exists \alpha_{i,q,\vec{q},J}\ :\ \bigwedge_{i,q,\vec{q},J} \vec{\Lambda}_{i,q,\vec{q},J} \in L_{i,q,\vec{q},J}^{\alpha_{i,q,\vec{q},J}} \wedge \bigwedge_{i=1}^{i=m} \Sigma_{q,\vec{q},J} \alpha_{i,q,\vec{q},J} > 0 \wedge \bigwedge_{l=1}^{l=n} \Psi_l(\alpha_l^1, \ldots, \alpha_l^{|Q_{N2AC}|})$$

```
for all q do  h=0; L_q^h = ∅ endfor
repeat
  for each q do  L_q^{h+1} = L_q^h endfor
  for each q do  case
      q ∈ Q_F then        repeat  add to L_q^{h+1} a multitree t = f(t_1, ..., t_n) s.t.
                                  t → q and ∀i = 1, ..., n, t_i ∈ L_{q_i}^h for some q_i
                          until   no new t can be added or |L_q^{h+1}| > B
      q ∈ Q_A then        repeat  add to L_q^{h+1} a multitree t = t_1. ... .t_m s.t.
                                  t_i ∈ L_{q_i}^h for some q_i
                          until   no new t can be added or |L_q^{h+1}| > B
      q ∈ Q_{N2AC} then   repeat  add to L_q^{h+1} a multitree t = n.t' s.t.
                                  t → q and n ≤ M, t' ∈ L_{q'}^h for some q'
                          until   no new t can be added or |L_q^{h+1}| > B
      q ∈ Q_{AC} then     repeat  add to L_q^{h+1} a multitree t = n_1.t_1 ⊕ ... ⊕ n_p.t_p s.t.
                                  t → q and n_i ≤ M less than B multitrees t_i
                                  reach the same state, t_i ∈ L_{q_i}^h for some q_i
                          until   no new t can be added or |L_q^{h+1}| > B
    esac
  h=h+1
until  there is some final state such that L^h(q) ≠ ∅ or L_q^h = L_q^{h+1} for all q's
if there is some final state q with L(q) ≠ ∅  then return non-empty
                                              else return empty
```

Figure 1: The emptiness decision algorithm

which is a Presburger's arithmetic formula Φ in the free variables $\Lambda^j_{i,q,\vec{q},J}$'s and α^k_l's (from proposition 3). The minimal solutions of $\exists \ldots \Lambda^j_{i,q,\vec{q},J} \cdots : \phi(\ldots, \Lambda^j_{i,q,\vec{q},J}, \ldots, \alpha^k_l, \ldots)$ give the minimal number of multitrees that much reach q_i in $Q_F \cup Q_A$ to get a multitree t reaching q^R_F using R as the last rule. Actually for each state $q_k \in Q_F \cup Q_A$, we need only $\alpha_k = Max_{l=1,\ldots,n}\alpha^k_l$ elements reaching q_k. Similarly, we need a bound on the coefficients of multitrees reaching $q_k \in Q_F \cup Q_A$, then we introduce $N_k = Max_{j=1,\ldots,n}(\Sigma_{i,\vec{q},J}\Lambda^j_{i,q_k,\vec{q},J})$ which bounds the maximal values of the coefficients of multitrees reaching q_k.

Finally we get the Presburger's arithmetic formula $\Psi(N_1, \ldots, N_p, \alpha_1, \ldots, \alpha_{|Q_{N2AC}|})$:

$$\exists \Lambda^j_{i,q,\vec{q},J}\ 's\ \exists \alpha^k_l\ 's \quad \begin{array}{l} \wedge \quad \Phi(\Lambda_{i,q,\vec{q},J}\ 's, \alpha_{i,q,\vec{q},J}\ 's) \\ \wedge \quad \bigwedge_{k=1}^{k=p} \alpha_k = Max_{l=1,\ldots,n}(\alpha^k_l) \\ \wedge \quad \bigwedge_{k=1}^{k=p} N_k = Max_{j=1,\ldots,n}(\Sigma_{i,\vec{q},J}\Lambda^j_{i,q_k,\vec{q},J}) \end{array}$$

Then we compute the set of minimal solutions of $\exists N_1, \ldots, N_p\ \Psi(N_1, \ldots, N_p, \alpha_1, \ldots, \alpha_{|Q_{N2AC}|})$ and a bound B_R which is the maximum of all components of these p-tuples. Moreover given some value B, for each minimal solution $(\alpha_1, \ldots, \alpha_{|Q_{N2AC}|})$ we can compute $M_R(B)$ defined by $M_R(B) = Max_B(\Psi(N_1, \ldots, N_p, \alpha_1, \ldots, \alpha_{|Q_{N2AC}|}))$.

8 Decidability of emptiness

We compute B the maximum of B_R for R a type 1or type 4 rule. Then we compute M the maximum of $M_R(B)$ for type $1, 3$ rules. The algorithm to decide emptiness is given in figure 1.

Theorem 1 follows from the next proposition.

Proposition 13 *The emptiness decision algorithm terminates and returns* **empty** *iff $\mathcal{L}(\mathcal{A})$ is empty.*

Proof Termination is obvious since at each step at least one $\mathcal{L}_q^h$ is augmented by one or more element, or no element is added to any $\mathcal{L}_q^h$.

The height $h(t)$ of a multitree t is defined by: $\begin{cases} h(f(t_1,\ldots,t_n)) = 1 + Max_{i=1,\ldots,n}(h(t_i)) \\ h(t_1.\ldots.t_m)) = 1 + Max_{i=1,\ldots,m}(h(t_i)) \\ h(\Sigma_i n_i t_i = 1 + Max_{i=1,\ldots,n}(h(t_i)) \\ h(n.t) = 1 + h(t) \end{cases}$

We define $L_q^h = \{t \mid t \to q \text{ } and \text{ } h(t) \leq h\}$. By construction, we have that $\mathcal{L}_q^h \subseteq L_q^h$. We show by induction on h that $\mathcal{L}_q^h = L_q^h$ or $|\mathcal{L}_q^h| > B$. We assume that the property holds for any $h' \leq h$ and we show that it holds for $h+1$. To this purpose, we discuss the way $\mathcal{L}_q^{h+1}$ is built, according to the type of the rule. The hypothesis that $\mathcal{A}$ is deterministic is used as follows: we construct new multitrees s_i's reaching a state from one multitree t reaching the same state by replacing a subterm u of t by other multitrees $v_1, v_2, \ldots$ also reaching the same state q than u. The determinism of $\mathcal{A}$ ensures that these v_i's are different from all other multitrees reaching another state q', therefore the s_i's are indeed new.

1. Type 1 rule.

   Let us assume that $t = f(t_1,\ldots,t_n) \to q$ using R as the last rule, where $t_i \in \mathcal{L}_{q_i}^h$. Moreover, we assume that t is minimal. There are three possibilities:

   (a) There is some q_i, say q_1, such that at least $B+1$ terms reach q_i. Then $\vec{t} = \vec{\lambda}_1 e_1 + \ldots + \vec{\lambda}_k e_k + \Sigma_{e'_j \to q_j, j>1} \vec{\lambda'}_j e'_j$ with $k \leq B$ (t is minimal) and $e_i \to q_1$ for $i = 1,\ldots,k$. Let $\_q_1{}^h = \{e_1,\ldots,e_k,\ldots,e_B,e_{B+1},\ldots\}$ be the multitrees reaching q_1. Since $k \leq B$, we can construct at least $C_{B+1}^B \geq B+1$ different multitrees multitrees reaching q using the same rule: in t replace $e_1,\ldots,e_k$ by $e_{i_1},\ldots,e_{i_k} \in \mathcal{L}_{q_1}^h$. Then $\mathcal{L}_q^{h+1}$ contains at least $B+1$ terms.

   (b) There are at least $B+1$ different sets of values for $\vec{\lambda_1},\ldots,\vec{\lambda}_{min}$ such that $\vec{t} = \vec{\lambda}_1 e_1 + \ldots + \vec{\lambda}_{min} e_{min} \to q$. Then $\mathcal{L}_q^{h+1}$ contains at least $B+1$ multitrees constructed from $\mathcal{L}_q^h$ and these sets of values.

   (c) None of the above cases holds. Then $\mathcal{L}_q^h = L_q^h$ and $\mathcal{L}_q^{h+1}$ contains all multitrees of height $h+1$ that can reach q using the rule R.

2. Type 2 rule. If there is a term $t_1. \ \ldots \ .t_m \in \mathcal{L}_{q_A}^{h+1}$ and $t_i \in \mathcal{L}_{q_i}^h$ where $|\mathcal{L}_{q_i}^h| > B$, tehn we can construct at least $B+1$ terms reaching q_A and $|\mathcal{L}_{q_A}^{h+1}| > B$.

3. Type 3 rule.

   Again, there are three possibilities:

   (a) If $|\mathcal{L}_q^h| > B$, then we can add at least $B+1$ multitrees to $\mathcal{L}_q^{h+1}$.

   (b) If $\phi(N)$ has at least $B+1$ solutions smaller than $\mathcal{M}$, then we add at least $B+1$ multitrees to $\mathcal{L}_q^{h+1}$.

   (c) If $\phi(N)$ has less than $B+1$ solutions smaller than $\mathcal{M}$, then all its solutions are smaller than $\mathcal{M}$. Since $\mathcal{L}_q^h = L_q^h$ in this case, $\mathcal{L}_q^{h+1}$ contains all multitrees of L_q^{h+1} reaching q using R as the last rule.

4. Type 4 rule.

   R is $\phi(N_1,\ldots,N_p): \ N_1 q_1 \oplus \ldots \oplus N_p q_p \to q$

   Let $t = \Sigma_{i=1}^{i=p} \Sigma_{s_i \to q_i} s_i$ be a minimal multitree reaching q.

   There are still three possibilities:

(a) Either there is some q_i, say q_1, such that at least $B+1$ independent multitrees reach q_1. We have $t = \Sigma_{i=1}^{i=k} s_i \oplus \Sigma_{i=2}^{i=p} \Sigma_{s'_i \to q_i} s_i$ with $k \leq B$ (t is minimal). Let $\mathcal{L}_{q_1}^h = \{s_1, \ldots, s_k, \ldots, s_B, s_{B+1}, \ldots\} \subseteq L_{q_1}^{h+1}$ be the set of terms reaching q_1 from multitrees of $L_{q'}^h$. We can chose at least $C_{B+1}^B \geq B+1$ distinct subsets of terms of S to build more than $B+1$ multitrees reaching q using the same rule: replace $s_1, \ldots, s_k$ by $s_{i_1}, \ldots, s_{i_k} \in S$ in t.

(b) There are at least $B+1$ possible solutions $\lambda_1, \ldots, \lambda_k, \ldots, \lambda_{B+1}, \ldots$ of $\phi(N)$ less than $\mathcal{M}$ for some rule, say $\phi(N) : Nq_1 \to q$. Therefore there are at least $C_{B+1}^B \geq B+1$ possible different subsets $\{\lambda_{i_1}, \ldots, \lambda_{i_k}\}$ since $k \leq B$. Each subset yield a different multitree reaching q using R as the last rule: replace in t the subterms $s_i = \lambda_i e_i$ by $\lambda_{j_i} e_i$ for $i = 1, \ldots, k$.

(c) None of the above case holds. Then $\mathcal{L}_q^{h+1}$ contains all the multitrees reaching q using R and multitrees of $\mathcal{L}_q^h$ (for $q \in Q_{NAC}$).

□

Theorem 1 *Let $\mathcal{A}$ be a deterministic constrained automaton, then it is decidable whether $\mathcal{L}(\mathcal{A})$ is empty or not.*

Since we can construct a deterministic automaton equivalent to a non-deterministic one, a straightforward consequence of theorem 1 is:

Theorem 2 *It is decidable whether $\mathcal{L}(\mathcal{A})$ is empty or not.*

An interesting question is whether the determinism is actually required to decide emptiness, since we mainly use the fact that we can build enough diferent multitrees from a large enough set of multitrees. The answer is probably no, but this is likely to add a lot of technicalities to the already technical proof.

9 A Query Language Based on a Tree Logic

As a first application of our new class of automata, we study query languages for semi-structured data [AB99], like XML documents. We take our inspiration from a recent proposal by Cardelli and Ghelli [CG01] that defines a query language for extensions of XML, *TQL*, itself based on a serendipitous connection with the ambient modal logic [CG00]. In this framework, a query is seen as a logical formula and the answers to a query are the models of the formula. Most particularly, model-checking amounts to finding all answers to a query. Following the same spirit, we design a logic for (multi)trees, *TL*, and show that satisfiability and model-checking are decidable problems in this logic. We do not aim at providing an extensive treatment of this subject, or to formally establish the relations between *TL* and *TQL* (we will need more space, but it will be the subject of a forthcoming work.) What we aim at, principally, is to prove the versatility and ease-of-use of our tree automata.

9.1 The Tree Logic TL

We consider multitrees constructed from a unique constant, 0, and a denumerable set of unary symbols, $\eta[\_]$, where η is primarily intended as a label. As previously, we also consider a concatenation and a (parallel) composition operator, $\cdot$ and $\oplus$. For simplicity reasons, we do not assume 0 to be neutral for $\oplus$, but this hypothesis can be safely added.

While the set of labels may be infinite, only a finite number of labels are used explicitly in a formula (or a database.) This allows to use co-finite sets, $\neq (\eta_1, \ldots, \eta_n)$, defining the infinite set of labels not in $\{\eta_1, \ldots, \eta_n\}$. While the finiteness of the alphabet is a usual assumption for regular tree or word languages, it is not a mandatory hypothesis for most of the properties and algorithms.

The language built on 0 and $\oplus$, $\cdot$ is a regular tree language that can be dealt with easily in our framework. It corresponds exactly to the set of terms considered in *TQL*. In this setting, a bibliographic database consisting of references labeled by *article* containing (unordered) fields for *author*, *title*, ... may be represented by a tree of the form (where η stands for the tree $\eta[0]$) :

$$\mathit{article}[\mathit{author}[\mathit{Knuth}] \oplus \mathit{author}[\mathit{Bendix}] \oplus \mathit{title}[\ldots] \oplus \ldots] \oplus \mathit{article}[\ldots] \oplus \ldots$$

Some examples of queries that a user could ask are: find *an article with no more than two authors*, find *an article written by Knuth and Bendix (no more authors) in 1970.* We could also look for articles which have duplicated identical fields (which probably indicate an erroneous entry.)

The syntax of Tree Logic formulas is given below. We use $A, B, \ldots$ to range over *TL*-formulas to avoid confusion with the Presburger's constraint of Section 4, and $\vec{N}$ and $\vec{A}$ for sequences of integer variables and formulas. For simplicity reasons, we identify a particular syntactic category of atomic formulas, $E, F, \ldots$, that corresponds to formulas over atomic terms $n\eta[\ldots]$.

$$\begin{array}{lll}
A, B, \ldots & ::= & \mathit{True} \mid 0 \mid \neg A \mid A \vee B \mid A \oplus B \mid \diamond A \mid \\
 & & \quad \mathit{Reg}_n(A_1, \ldots, A_n) \mid \exists N.\psi(N) : N\eta[A] \mid \exists \vec{N}.\psi(\vec{N}) : N_1 E_1 \oplus \ldots \oplus N_p E_p \\
E, F, \ldots & ::= & \exists N.\psi(N) : N\eta[A]
\end{array}$$

This logic resembles the (static part) of the ambient logic. We have a logical constant for the empty tree, 0. We have a tensor product, $A \oplus B$, denoting trees where A and B are satisfied "contiguously" (this is noted $A|B$ in [CG00].) And we have the *eventually* modality, $\diamond A$, denoting that there is a subtree satisfying A.

The last three operators of *TL* are original. The formula $\mathit{Reg}_n(A_1, \ldots, A_n)$ defines a regular expression over the alphabet $A_1, \ldots, A_n$, for example $(A_1 + A_2) \cdot A_3^*$, and allows to express recursive properties over paths along $\cdot$. For simplicity reasons, we consider that the A_i's are not of the form $\mathit{Reg}_k(\ldots)$. This modality corresponds to type 2 rules and allows expressing queries of the kind provided by hedge automata [Mur01]. The extended existential quantification correspond to type 3 and type 4 rules. For the sake of brevity, we use $\bigoplus \vec{N}\vec{E}$ to denote the product $N_1 E_1 \oplus \cdots \oplus N_p E_p$. Then, the formula $\exists \vec{N}.\psi(\vec{N}) : \bigoplus \vec{N}\vec{E}$ denotes trees of the form $\bigoplus n_i t_i$ where t_i satisfies E_i for all $i \in 1..p$ and $\psi(n_1, \ldots, n_p)$ holds. Note that the syntax force the E_i's to be atomic formulas, that is t_i to be of the form $n\eta[\ldots]$.

We define the satisfaction relation $t \models A$, meaning that the tree t satisfies A.

$$\begin{array}{llcl}
\forall t. & t \models \mathit{True} & & \text{(always true)} \\
\forall t, A. & t \models \neg A & \triangleq & \neg\, t \models A \\
\forall t, A, B. & t \models A \vee B & \triangleq & t \models A \ \vee\ t \models B \\
\forall t. & t \models 0 & \triangleq & t \equiv 0 \\
\forall t, A, B. & t \models A \oplus B & \triangleq & \exists s, s'.\ t \equiv s \oplus s' \ \wedge\ s \models A \ \wedge\ s' \models A' \\
\forall t, A. & t \models \diamond A & \triangleq & t \models A \\
 & & & \vee\ (t = \eta[t'] \vee t = nt') \wedge t' \models \diamond A \\
 & & & \vee\ t = n_1 t_1 \oplus \ldots \oplus n_p t_p \wedge \exists i \in 1..p. t_i \models A \\
\forall t, \vec{A}. & t \models \mathit{Reg}_n(A_1, \ldots, A_n) & \triangleq & t = t_1 \cdot \ldots \cdot t_p \ \wedge\ \forall i \in 1..p\ \exists j_i \in 1..n.\ t_i \models A_{j_i} \\
 & & & \ \wedge\ A_{j_1} \cdot \ldots \cdot A_{j_m} \in \mathcal{L}(\mathit{Reg}_n(A_1, \ldots, A_n)) \\
\forall t, A. & t \models \exists N.\psi(N) : N\eta[A] & \triangleq & t = n\eta[t'] \text{ where } t' \models \phi \ \wedge\ \models \psi(n) \\
\forall t, \vec{N}, \vec{E}. & t \models \exists N.\psi(\vec{N}) : \bigoplus \vec{N}\vec{E} & \triangleq & t \equiv \sum_{i \leqslant p} m_i^1 t_i^1 \oplus \cdots \oplus m_i^{n_i} t_i^{n_i} \\
 & & & \ \wedge\ \forall j \in 1..n_i.\ m_i^j t_i^j \models E_i \ \wedge\ \models \psi(n_1, \ldots, n_p)
\end{array}$$

This is a crude logic and a realistic proposition requires some syntactic sugar to be more user friendly, as was done in [CG01]. Indeed, it is more appropriate to understand the modalities in *TL* as a kind of abstract machine operations underlying a more complex query language. Nonetheless, it is possible to give some simple examples of interesting queries. For example, $X \models \mathit{article}[A] \oplus \mathit{True}$ means that the tree X contains an article satisfying some property A (and maybe something else). We write this kind of formulas $(X \models (\mathit{article}[Y] \oplus \mathit{True}))$ **where** $(Y \models A)$ to get more readable expressions latter. Another example is a formula denoting articles with two identical *author* fields

$$\exists N, M.(N \geqslant 1) : N Z \oplus M(\neq \mathit{author})[\mathit{True}] \; \textbf{where} \, (Z \models \exists N.N \geqslant 2 : N \mathit{author}[\mathit{True}])$$

The following example denotes articles written in 1970 by at most two different authors:

$$\exists M, N, P.(M \leqslant 2) \wedge (N = 1) : M \mathit{author}[\mathit{True}] \oplus N \mathit{year}[1970] \oplus P \neq (\mathit{year}, \mathit{author})[\mathit{True}]$$

As a last example, we can define a formula denoting articles written by Knuth and Bendix in 1970. It is enough to take the conjunction of the two previous formulas together with the one given below (but it is also possible to build a more direct and efficient query):

$$\begin{array}{rcll} \exists M, N, P.(M = 1) \wedge (N = 1) : & & M & (\exists N.(N = 1) : N \mathit{author}[\mathit{Knuth}]) \\ & \oplus & N & (\exists N.(N = 1) : N \mathit{author}[\mathit{Bendix}]) \\ & \oplus & P & (\neq (\mathit{year}, \mathit{author})[\mathit{True}]) \end{array}$$

9.2 Decidability Results and Extensions

Since modalities of *TL* directly correspond to operations on multitree automata, we can easily lift Theorem 2, our decidability results on recognized languages, to the level of the logic. Indeed, there is a direct and simple interpretation of formulas as automata that to any formula, A, associates an automaton, $\mathcal{A}$, such that the language recognized by $\mathcal{A}$ is exactly the set $\{t \,|\, t \models A\}$ of trees satisfying A. In this setting, it is equivalent to say that A is satisfiable or that $\mathcal{L}(\mathcal{A})$ is non-empty. The result for model-checking is even simpler since, by definition, $t \models A$ is equivalent to $\mathcal{A}$ accepts t. More than a result on the complexity of the logic, what we obtain is in fact an effective way to model-check and test the satisfiability of formulas.

Theorem 3 *Satisfiability and model checking are decidable for the tree logic TL.*

Proof The proof follows the usual pattern: to each formula we associate an automaton which recognizes the models of the formula. This is done by structural induction on the formula which reflects the closure property of multitree automata and the design of the logic.

- The cases of 0 or $True$ are straightforward.
- The case of $\neg\varphi, \varphi \vee \varphi, \varphi \oplus \varphi$ directly follow from the closure properties of section 6.
- The case of $Reg_n(\varphi_1, \ldots, \varphi_n)$ is simple. For each φ_i we assume that there is an automaton $\mathcal{A}_i$ recognizing the models of φ_i. We take the union of these automata add a new state which is the unique final state q_φ and the rule $L(\mathbf{A}_\varphi) \to q_\varphi$ where $\mathbf{A}_\varphi$ is the automaton accepting the language generated by the regular expressions $q_{\varphi_1}. \ \ldots . q_{\varphi_n}$ where the q_φ are final state of $|A_i$ or a regular expression generating $L(\mathbf{A}_i)$ if $\mathcal{A}_i$ contains the rule $L(\mathbf{A}_i) \to q_A^{final}$ with q_A^{final} a final state of $\mathcal{A}_i$.
- The case $\psi(N) : N\eta[\varphi]$ is not difficult. We take $\mathcal{A}_\varphi$ an automaton accepting the models of φ, set the final states to $\{q_{success}\}$ ($q_{success}$ a new state) and we add the rules

$$\begin{array}{l} True : \eta(q_\varphi \to q_\varphi^F \text{ with } q_\varphi \text{ a final state of } \mathcal{A}_\varphi,\ q_\varphi^F \text{ a new state} \\ \psi(N) : N q_{varphi}^F \to q_\varphi^{N2AC} \text{ with } q_\varphi^{N2AC} \text{ a new state} \\ N = 1 \wedge \bigwedge_{i \geq 1} N_i = 0 : N q_\varphi^{N2AC} \to q_{success} \oplus \Sigma_{i \geq 1} N_i q_i \end{array}$$

- The case of $\psi(N_1, \ldots, N_p) : N_1\varphi_1 \oplus \ldots \oplus N_p\varphi_p$ is easy when the φ_i's are pairwise unsatisfiable (which is usually true). In this case, we take the union of the automata accepting φ_i, set the set of final states to $\{q_{success}\}$ and we add the rules:

$$\psi(N_1, \ldots, N_p) : N_1 q_{\varphi_1} \oplus \ldots \oplus N_p q_{\varphi_p} \to q_{success}$$

By construction if a term is accepted by the automaton it satisfies the initial formula. Conversely, a term satisfying the formula is necessarily $t = \underbrace{t_1^1 \oplus t_{n_1}^1}_{t_j^1 \models \varphi_1} \oplus \ldots \oplus \underbrace{t_1^1 \oplus t_{n_p}^1}_{t_j^p \models \varphi_p}$ where each t_j^i is some $n_j^i s_j^i$. Since t_j^i satisfies only φ_i, we have necessarily $\models \psi(n_1, \ldots, n_p)$, then t is accepted.

In the general case, we perform some kind of determinization of formulas. We consider all possible partitions of $\{1, \ldots, p\}$. Let $J_1, \ldots, J_m$ be one such partition, then we set ϕ_l for $l = 1, \ldots, m$ to be $\bigwedge_{i \in J_l} \varphi_i \wedge \bigwedge_{i \notin J_l} \neg\varphi_i$. By construction the ϕ_i's are pairwise unsatisfiable. Now consider let $\Psi_{J_1,\ldots,J_l}$ the formula

$$\exists x_i^k \quad \textstyle\bigwedge_{i=1}^{i=l} M_i = \Sigma_{k \in J_i} x_k^i : \bigwedge_{j=1}^{j=p} N_j = \Sigma_{i=1}^{i=l} x_j^i \wedge \psi(N_1, \ldots, N_p) : \; M_1\phi_1 \oplus \ldots \oplus M_p\phi_p$$

Then the initial formula is equivalent to $\bigvee_{\{J_1,\ldots,J_l\} partition of \{1,\ldots,p\}} \Psi_{J_1,\ldots,J_l}$. By construction the models of these formula are included in the model of the original formula. Conversely for each $t = t_1 \oplus \ldots \oplus t_n$ satisfying the formula, we group together the t_i's satisfying the same φ_i's and we get that t satisfies one of the above $\Psi_{J_1,\ldots,J_l}$.

- The case $\Diamond\varphi$ requires little work. Take an automaton accepting φ add new final states $\{q_s, q_A, q_{s,AC}\}$ and the rules

$$\begin{array}{l} True : \eta(\_) \to q_s \text{ if \_ is any final state} \\ N \geq 1 : N\_ \to q_{s,N2AC} \text{ if \_ is any final state} \\ N \geq 1 : Nq_{s,N2AC} \oplus \ldots \to q_{s,AC} \end{array}$$

$\square$

By looking at the definition of *TL*, it is possible to define a simplification of multitree automata that still enjoys similar simulation results. This simplification is obtained by replacing type 3 and type 4 rules with rules of the form $\psi(\vec{N}) : N_1q_1 \oplus \ldots \oplus N_pq_p \to q$, and modifying the relation $\to_{\mathcal{A}}$ accordingly: there is a transition $t \to_{\mathcal{A}} q$ if $t = \sum_k n_k t_k$ and $\models \psi(m_1, \ldots, m_p)$ where $m_i = \sum\{n_k \,|\, t_k \to q_i\}$. Unfortunately this natural definition forbids the determinization of automata, and it is not even clear whether the intersection of two recognizable languages is still recognizable. This provides another example of the fact that, while it may be easy to devise new classes of tree automata, it is much more difficult to find classes that enjoy all the good typical properties.

There exists many possible extensions to this logic. The crucial point is that we can use our underlying tree-automata model to easily add new modalities to *TL* while still preserving the decidability results of Theorem 3. For instance, we could add some typical temporal logic modalities, like A **until** B. Since our structures are finite, we do not need all the power of automata on infinite structures as it is the case with temporal logics. This modality allows to get regular expressions for paths along $\oplus$. For example, it permits to check that the tree $a[b[0] \oplus a[c[0]]]$ has a path of the form $a^+c[0]$. Another possible extension is to add monadic second-order variables and quantification on these variables, $A ::= \ldots \;|\; \mathcal{X} \;|\; \forall\mathcal{X}.A$. Then, as usual, a second-order variable is interpreted by a set of multitrees. Unfortunately the resulting logic is undecidable (by encoding two counter machines) but the uniform fragment is decidable because it enjoys a monotonicity property. Here uniform means that each variable occurs only positively or occurs only negatively, where positive/negative occurrences of a variable are defined as usual. Therefore, to check satisfiability, it suffices to replace positive variables by the empty set (equivalently $\neg$ *True*), and negative variables by the set of all multitrees (that is *True*).

10 An application to inductive theorem proving

The *inductionles induction* method [JK89] is a powerful approach in automated theorem proving which often succeeds when methods based on structural induction fail. In this framework, the axioms are equational axioms which are oriented into rewrite rules and a completion-like process computes inductive consequences of the theorem to prove until a contradiction is derived or all subgoals are proved. The key property to check is that a term is *inductively reducible* [Pla85], i.e. all its ground instances are reducible by the rewrite system. This method has been extended to deal with non-orientable axioms like associativity-commutativity. Unfortunately, inductive reducibility is undecidable in the associative-commutative case [KNRZ91]. However the linear case is decidable, as well as also some non-linear cases [LM94]. We briefly describe how we can use multitree automata and (a slight extension of) the modal logic TL to extend the decidable cases. From now on, we consider multitrees build on $\mathcal{F} = F \cup \{\oplus\}$ and a set $\mathcal{X}$ of variables (assuming more than one associative-commutative symbol doesn't raise any difficulty). Ground multitrees are multitrees without variables. The instances of a multitree are the multitree obtained by replacing the variables by any ground multitree.

A multitree t (with variables) has non-linearity restricted to brother iff

- either $t = f(t_1, \ldots, t_n)$ where (i) $t_i = x_i$ implies if x_i occurs elsewhere in t then x_i is some t_j, and (ii) for all i, t_i has non-linearity restricted to brothers,
- or $t = \Sigma_k n_i t_i$ with $n > 1$ where (i) if t_i is a variable x then x doesn't occur in the other t_j's and (ii) for all i, t_i has non-linearity restricted to brothers .

A term t is inductively reducible with respect to a set of terms $R = \{t_1, \ldots, t_n\}$ iff any ground instance of t contains a ground instance of some t_i.

Proposition 14 *The inductive reducibility of a term t with respect to a set of terms $R = \{t_1, \ldots, t_n\}$ where the non-linearity is restricted to brothers is decidable.*

Proof We show that inductive reducibility can be expressed in TL (extended to a signature with free symbols of any arity, and with the corresponding type 4 rules).

Claim: *Let t be a term with restricted non-linearity. Then the set of ground instances of t is the model of a TL formula $instance(t)$.*

The proof is by structural induction on t.

- t is a variable x. Then $instance(t)$ is $True$.
- $t = nt'$, where t' is some $f(\ldots)$. Then s is a ground instance of t iff $\models (N = n) : N\ instance(t')$.
- $t = s_1 \oplus \ldots \oplus s_p$ with $s_i = n_i t_i$. Then s is a ground instance of t iff $s \models instance(s_1) \oplus \ldots \oplus instance(s_p)$
- $t = f(t_1, \ldots, t_n)$ then $instances(t)$ is $\psi(X_1, \ldots, X_n) : f(instances(t_1), \ldots, instances(t_n))$ where $\psi(X_1, \ldots, X_n) = \bigwedge_{i,j \mid t_i = t_j \in \mathcal{X}} X_i = X_j$.

Claim: *Let t be a term with restricted non-linearity. Then the set of ground terms containing an instance of t is the model of a TL formula.*

Let $instance(t)$ be the TL formula describing the ground instances of t. Then $\Diamond\varphi_t$ describes the ground terms containing an instance of t.

From the two previous claims we get the formula expressing that t is not inductively reducible:

$$instance(t) \wedge \neg(\Diamond instance(l_1) \vee \ldots \vee \Diamond instance(l_n))$$

This formula is satisfiable if there is an instance of t which is not an instance of any l_i.

□

The complexity of this decision procedure is too high for practical purposes. But the main point here is to have a closer look at the frontier between decidable and undecidable cases.

Conclusion

We have presented and studied multitree automata and a related decidable tree logic. This work can be continued in many directions. A first route is to develop the tree logic. We have already proposed several extensions: new modalities, second-order variables, ... and we still need to explore the complexity of these extensions. In fact, our tree logic uses only a restricted portion of the power of multitree automata. Most particularly, since all functions are either constant or monadic symbols, Presburger's constraints (used in type 1 rules) are quite simple. Therefore we can hope for a better complexity than the one arising from our algorithm to decide emptiness. But we should not hope too much, in the presence of AC symbols, even the matching problem is NP-complete.

Another route is to investigate different kind of multitree structures. Indeed, it can sometimes be useful to use a multiset representation of trees, rather than multitrees where identical terms are grouped together, using $a \oplus b \oplus a$ instead of $2a \oplus b$ for instance. In this case, we have also designed tree automata that exhibit good properties. Note that in this case, while we still have type 4 and type 1 rules and that some counting ability remains, we loose the ability to have type 3 rules. The benefit of this approach is that regular tree languages fall into this class as soon as they are closed under associativity-commutativity.

From a pure logical point of view, we should mention that it is common to present transition rules of tree automata as (Horn) clauses. In this framework, we can easily design alternating tree automata, meaning that we can go up or down a tree during the acceptance process. Goubault and Verma [GLV02] have recently used this approach to find a new decidable class of clauses with additional associative-commutative axioms. It would be interesting to see if multitree automata give some hint at a possible extension of this result.

References

[AB99] S. Abiteboul and P. Buneman. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[AW92] A. Aiken and E. Wimmers. Solving systems of set constraints. In *Proc. 7 th IEEE Symp. on Logic in Computer Science, Santa Cruz*, pages 329–340, 1992.

[BT92] B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In *Proceedings of the 9th Symposium on Theoretical Computer Science*, volume 577 of *Lecture Notes in Computer Science*, pages 161–172, 1992.

[CCC+94] A.C. Caron, H. Comon, J.L. Coquidé, M. Dauchet, and F. Jacquemard. Pumping, cleaning and symbolic constraints solving. In *Proceedings 21st ICALP Conference, Jerusalem (Israel)*, pages 436–449, 1994.

[CCD93] A-C. Caron, J-L. Coquide, and M. Dauchet. Encompassment properties and automata with constraints. In C. Kirchner, editor, *Proceedings 5th International Conference on Rewriting Techniques and Applications (Montreal, Canada)*, volume 690 of *Lecture Notes in Computer Science*, pages 328–342. Springer-Verlag, 1993.

[CCM01] H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints and ping-pong protocols. In *Proceedings of 28th Int. Coll. Automata, Languages, and Programming (ICALP'2001)*, volume 2076 of *Lecture Notes in Computer Science*, pages 682–693. Springer-Verlag, 2001.

[CG00] L. Cardelli and A. Gordon. Anytime, anywhere: Modal logic for mobile ambients. In *Proc. of Principles of Programming Languages (POPL)*. ACM, January 2000.

[CG01] L. Cardelli and G. Ghelli. A query language based on the ambient logic. In *Proceedings of ESOP'01*, volume 2028 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, 2001.

[CJ94] H. Comon and F. Jacquemard. Ground reducibility and automata with disequality constraints. In Springer-Verlag, editor, *Proceedings of 11th Symposium on Theoretical Computer Science STACS*, number 820 in Lecture Notes in Computer Science, pages 151–162, 1994.

[Com00] H. Comon. Sequentiality, monadic second-order logic and tree automata. *Information and Computation*, 157(1-2):25–51, 2000.

[DJ90] N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990.

[Don70] J.E. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.

[GLV02] J. Goubault-Larrecq and K.N. Verma. Alternating two-way AC-tree automata. Technical report, LSV, ENS Cachan, 2002.

[JK89] J.P. Jouannaud and E. Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82(1), 1989.

[KNRZ91] D. Kapur, P. Narendran, D. J. Rosenkrantz, and H. Zhang. Sufficient completeness, ground-reducibility and their complexity. *Acta Informatica*, 28:311–350, 1991.

[LM94] D. Lugiez and J.L. Moysset. Tree automata help one to solve equational formulae in AC-theories. *Journal of Symbolic Computation*, 18(4):297–318, 1994.

[Lug98] D. Lugiez. A good class of tree automata. In K. Larsen, Sven Skyum, and G. Winskel, editors, *ICALP 98*, number 1443 in Lecture Notes in Computer Science, pages 409–420. Springer-Verlag, July 1998.

[Mur01] Makoto Murata. Extended path expression for XML. In ACM, editor, *Proceedings of the Twenteenth Symposium on Principles of Database Systems (PODS)*, Santa Barbara, USA, 2001. ACM.

[NP93] Joachim Niehren and Andreas Podelski. Feature automata and recognizable sets of feature trees. In *Proceedings TAPSOFT'93*, volume 668 of *Lecture Notes in Computer Science*, pages 356–375, 1993.

[Ohs01] Hitoshi Ohsaki. Beyond the regularity: Equational tree automata for associative and commutative theories. In *Proceedings of CSL 2001*, volume 2142 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[OP99] P. O'Hearn and D. J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

[Pla85] D. Plaisted. Semantic confluence and completion method. *Information and Control*, 65:182–215, 1985.

[PQ68] C. Pair and A. Quéré. Définition et étude des bilangages réguliers. *Information and Control*, 13(6):565–593, 1968.

[Rab77] M.O. Rabin. *Handbook of Mathematical Logic*, chapter Decidable Theories, pages 595–627. North-Holland, 1977.

[TW68] J.W. Thatcher and J.B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–82, 1968.

[Var97] Moshe Y. Vardi. Alternating automata: Unifying truth and validity checking for temporal logics. In William McCune, editor, *Proceedings of the 14th International Conference on Automated deduction*, volume 1249, pages 191–206, Berlin, 13–17 1997. Springer.