Tree automata help one to solve equational formulae in AC-theories

Denis Lugiez, Jean-Luc Moysset

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

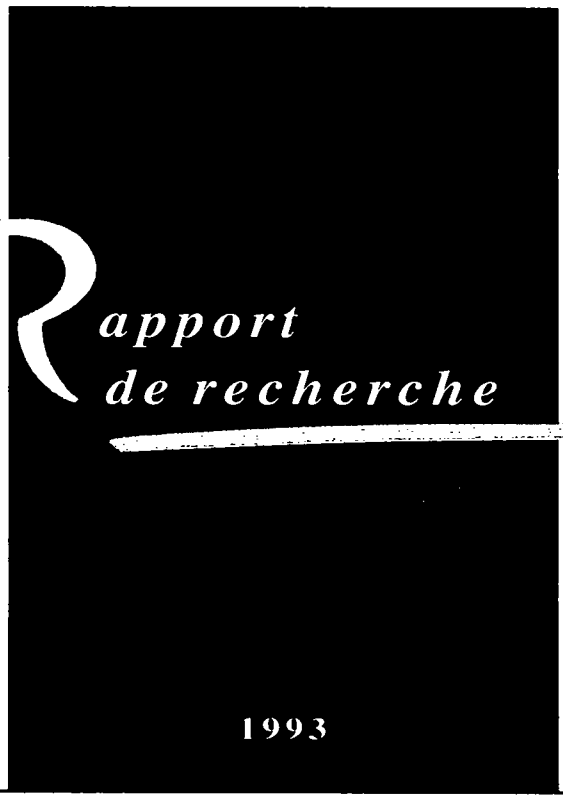
Tree Automata Help one to Solve Equational Formulae in AC-Theories

Denis LUGIEZ
Jean-Luc MOYSSET

N° 2029
Septembre 1993

PROGRAMME 2

Calcul symbolique,
programmation
et génie logiciel



Les automates d'arbres aident à la résolution des formules équationnelles dans les théories AC

Denis Lugiez, Jean-Luc Moysset

Résumé

Dans cet article, nous étudions des formules équationnelles particulières où l'égalité $=_{AC}$ est la congruence engendrée par un ensemble d'axiomes d'associativité et commutativité. Les formules auxquelles nous nous intéressons sont de la forme $t \neq_{AC} t_1 \wedge \ldots t \neq_{AC} t_n$ et sont habituellement appelées *problème de compléments*. Résoudre un problème de complément revient à trouver une instance de t qui n'est instance d'aucun t_i modulo l'associativité-commutativité. Nous donnons une méthode de décision basée sur les automates d'arbres qui résoud le problème quand tous les t_i sont linéaires. Nous montrons que cette approche donne une méthode de décision de la réductibilité inductive pour les systèmes linéaire gauches et nous décrivons plusieurs extensions de notre approche. Puis nous définissons une nouvelle classe d'automates d'arbres qui reconnait des ensembles de termes normalisés (i.e écrits sous forme de multi-ensemble) pour lesquels l'application d'une règle dépend d'une formule de l'arithmétique de Presburger. Cette classe d'automates d'arbres permet de résoudre les problèmes de compléments non-linéaires si toutes les occurrences d'une variable non-linéaire sont sous le même noeud (dans les termes aplatis). Cette solution donne aussi une méthode de décision de la réductibilité inductive dans ce cas.

Tree Automata help one to solve equational formulae in AC-theories

Abstract

In this paper we consider particular equational formulae where the equality $=_{AC}$ is the congruence induced by a set of associative-commutative axioms. The formulae we are interested in have the form $t \neq_{AC} t_1 \wedge \ldots t \neq_{AC} t_n$ and are usually known as complement problems. To solve a complement problem is to find an instance of t which is not an instance of any t_i modulo associativity-commutativity. We give a decision procedure based on tree automata which solves these formulae when all the t_i 's are linear. We show that this approach gives a decision of inductive reducibility modulo associativity and commutativity in the linear case and we give several extensions of this approach. Then, we define a new class of tree automata, called conditional tree automata which recognize sets of normalized terms (i.e terms written as multisets) and where the application of a rule depends on the satisfiability of a formulae of Presburger's arithmetic. This class of tree automata allows one to solve non-linear complement problems when all occurrences of a non-linear variable occur under the same node (in flattened terms). This solution also provides a decision of the inductive reducibility modulo associativity-commutativity in the same case.

Les automates d'arbres aident à la résolution des formules équationnelles dans les théories AC

Denis Lugiez, Jean-Luc Moysset

Résumé

Dans cet article, nous étudions des formules équationnelles particulières où l'égalité $=_{AC}$ est la congruence engendrée par un ensemble d'axiomes d'associativité et commutativité. Les formules auxquelles nous nous intéressons sont de la forme $t \neq_{AC} t_1 \wedge \ldots t \neq_{AC} t_n$ et sont habituellement appelées *problème de compléments*. Résoudre un problème de complément revient à trouver une instance de t qui n'est instance d'aucun t_i modulo l'associativité-commutativité. Nous donnons une méthode de décision basée sur les automates d'arbres qui résoud le problème quand tous les t_i sont linéaires. Nous montrons que cette approche donne une méthode de décision de la réductibilité inductive pour les systèmes linéaire gauches et nous décrivons plusieurs extensions de notre approche. Puis nous définissons une nouvelle classe d'automates d'arbres qui reconnait des ensembles de termes normalisés (i.e écrits sous forme de multi-ensemble) pour lesquels l'application d'une règle dépend d'une formule de l'arithmétique de Presburger. Cette classe d'automates d'arbres permet de résoudre les problèmes de compléments non-linéaires si toutes les occurrences d'une variable non-linéaire sont sous le même noeud (dans les termes aplatis). Cette solution donne aussi une méthode de décision de la réductibilité inductive dans ce cas.

Tree Automata help one to solve equational formulae in AC-theories

Abstract

In this paper we consider particular equational formulae where the equality $=_{AC}$ is the congruence induced by a set of associative-commutative axioms. The formulae we are interested in have the form $t \neq_{AC} t_1 \wedge \ldots t \neq_{AC} t_n$ and are usually known as complement problems. To solve a complement problem is to find an instance of t which is not an instance of any t_i modulo associativity-commutativity. We give a decision procedure based on tree automata which solves these formulae when all the t_i 's are linear. We show that this approach gives a decision of inductive reducibility modulo associativity and commutativity in the linear case and we give several extensions of this approach. Then, we define a new class of tree automata, called conditional tree automata which recognize sets of normalized terms (i.e terms written as multisets) and where the application of a rule depends on the satisfiability of a formulae of Presburger's arithmetic. This class of tree automata allows one to solve non-linear complement problems when all occurrences of a non-linear variable occur under the same node (in flattened terms). This solution also provides a decision of the inductive reducibility modulo associativity-commutativity in the same case.

Tree Automata help one to solve equational formulae in AC-theories

D.Lugiez, J.L. Moysset
*
CRIN-INRIA Nancy FRANCE

key words: Automata and Formal Languages, Logic in Computer Science, Automated Theorem Proving, Program specification, Equational Theory.

Abstract

In this paper we consider particular equational formulae where the equality $=_{AC}$ is the congruence induced by a set of associative-commutative axioms. The formulae we are interested in have the form $t \neq_{AC} t_1 \wedge \ldots t \neq_{AC} t_n$ and are usually known as complement problems. To solve a complement problem is to find an instance of t which is not an instance of any t_i modulo associativity-commutativity. We give a decision procedure based on tree automata which solves these formulae when all the t_i 's are linear. We show that this approach gives a decision of inductive reducibility modulo associativity and commutativity in the linear case and we give several extensions of this approach. Then, we define a new class of tree automata, called conditional tree automata which recognize sets of normalized terms (i.e terms written as multisets) and where the application of a rule depends on the satisfiability of a formulae of Presburger's arithmetic. This class of tree automata allows one to solve non-linear complement problems when all occurrences of a non-linear variable occur under the same node (in flattened terms). This solution also provides a decision of the inductive reducibility modulo associativity-commutativity in the same case.

Introduction

Many problems arising in Computer Science involve formulae built on the equality predicate and syntactic objects, usually terms. But terms are merely denotations for complex entities and a purely syntactical approach lacks any insight on the semantics of these entities. To avoid this drawback, a classical solution is to add axioms on terms which model the semantic behavior of the real objects. Amongst

*author's address: e-mail lugiez,moyssetj@loria.fr, surface-mail CRIN-INRIA, BP239 54506 Vandoeuvres-les-Nancy, FRANCE

the most useful axioms one finds the *commutativity* axiom (C) $f(x,y) = f(y,x)$, the *associativity* axiom (A) $f(x,f(y,z)) = f(f(x,y),z)$, the *idempotency* axiom (I) $f(x,x) = x$ and the *unit element* axiom (1) $f(x,e) = x$. Many operators studied in Computer Science satisfy one or more of these axioms, for example the parallel operator in parallelism is associative-commutative (AC in short), the boolean connectives are associative-commutative and idempotent (ACI in short), and many algebraic functions are associative-commutative and have a unit element (AC1 in short). Therefore we are led to tackle the problem of dealing with formulae on equality modulo a set of axioms. Decision procedures exist when no axiom is included [Mal71, Col84, LM86, Mah88, CL89] but a decision procedure for these formulae usually does not exist in presence of axioms (indeed, when AC-axioms occur, the theory of Σ_3 formulae is undecidable [Tre92]). Actually, we are asking for too much: in practice, the formulae that we must solve belong to very restricted classes. After the well-known unification problem, the most interesting one is that of *complement problem*, i.e to solve $t \neq_E t_1 \wedge \ldots \wedge t \neq_E t_n$ where E is the theory we are interested in. This kind of formulae occur in algebraic specification when dealing with sufficient completeness, in logic programming and constraint logic programming, especially concerning constructive negation, in functional programs to decide the completeness of a function definition and in the paradigm of learning by example and counter-example in the field of machine learning (see [LMK91] for example).

This paper describes a new approach to the AC-complement problem which relies on tree automata. First, we show how tree automata solve linear AC-complement problems and we give some extensions of this result. Then we describe a new class of tree automata, called conditional tree automata which works on (some kind of) flattened trees and involves conditions which are formulas of Presburger's arithmetic. Using this new class, we can solve a non-linear case of the AC-complement problem, more precisely when all the occurrences of a non-linear variable are under the same AC-symbol. Moreover our approach can be extended to the decision of the inductive reducibility property modulo AC which is a key notion in inductionless induction. This property is undecidable [KNRZ91], but we are able to decide it in the aforementioned cases.

This paper is organized as follows: section one deals with definitions and notations, section 2 describes the linear case and section 3 provides the easiest extensions of the linear case. In section 4, we introduce our new class of tree automata which is used in section 5 to give a solution to the so-called restricted non-linear case.

1 Definitions and notations

We need some definitions and notations, let us start with terms. Missing definitions can be found in [DJ90].

1.1 Terms

Terms, denoted by $s,t,r,\ldots$ are labelled trees constructed from a finite set F of function symbols denoted by $f,g,h,\ldots$, and a denumerable set X of variables, denoted $x,y,z,\ldots$. Functions of arity 0 are constants. We suppose that F contains at least

one constant, therefore the set of ground terms, i.e. terms without variables is not empty. A term t is *linear* if each variable occurring in t occurs only once in t. $Var(t)$ is the set of variables occurring in t. A *position* is a sequence of integers, the empty sequence is denoted by ϵ, and a non-empty sequence is written as $p.i$ with i an integer and p a sequence. The subterm of a term s at position p, denoted by $s_{|p}$, is defined by $s_{|\epsilon} = s$ and $s_{|p.i} = s_i$ if $s_{|p} = f(s_1, \ldots, s_n)$ and $1 \leq i \leq n$. A subterm of s is *strict* if it is different from s. The *arguments* of a term $s = f(s_1, \ldots, s_n)$ are the s_i's. A *context* $C[.]$ is a term with one hole and $C[s]$ is the term obtained by putting the term s in the hole of the context $C[.]$.

An equation (i.e. an axiom) is a pair of terms written $s = t$. A finite set of equations E defines an equational theory and induces a congruence relation on terms denoted by $=_E$. The class of a term t is the set of terms equal to t modulo E. We are mainly interested in the following equations: commutativity $f(x,y) = f(y,x)$, associativity $f(x, f(y,z)) = f(f(x,y), z)$, idempotency $f(x,x) = x$ and unit element $f(x,e) = x$. More precisely we will consider the congruences $=_{AC}$ (resp. $=_{ACI}, \ldots$) i.e $F = F_{AC} \cup F_{NAC}$ (resp. $F = F_{ACI} \cup F_{NACI}, \ldots$) where all function symbols in F_{AC} are commutative and associative (resp. and idempotent...) and the function symbols in F_{NAC} are free.

Substitutions, denoted by $\sigma, \theta, \rho \ldots$, are the morphisms on terms and the application of σ to t is denoted by $t\sigma$. The domain of a substitution σ is the set $Dom(\sigma) = \{x \in X; x\sigma \neq x\}$. A term s is an E-instance of a term t iff $s =_E t\sigma$. It is a ground E-instance if s is ground. When E is the empty theory, we shall of instances and ground instances, dropping the E prefix.

To solve the **complement problem** $t \neq_E t_1 \wedge \ldots \wedge t \neq_E t_n$ modulo E with $\vec{x} \cap \vec{y_i} = \emptyset$ and $\vec{y_i} \cap \vec{y_j} = \emptyset$ if $\vec{y_i} = Var(t_i)$ and $\vec{x} = Var(t)$, is to find a ground instance of t which is not a E-instance of any t_i. In other words, it amounts to deciding the validity of the formula $\exists \vec{x} \forall \vec{y_1} \ldots \vec{y_n} : t \neq_E t_1 \wedge \ldots \wedge t \neq_E t_n$ in the algebra of ground terms.

A term rewrite system R is a finite set of rewrite rules $l \rightarrow r$ which defines a rewrite relation $\rightarrow_R$ on terms. The system is left-linear if for each rule $l \rightarrow r$ of R, l is linear. A term t is reducible by R modulo AC iff there are some term t', position p in t', substitution θ and term $l \in R$ s.t. $t' =_{AC} t$ and $t'_{|p} = l\theta$. A key property in inductionless induction is that of *inductive reducibility*: given a set E of equations and a set R of rewrite rules, a term t is inductively reducible iff each ground instance of t is reducible. This property is undecidable for the AC theory, but our solution to linear complement problems provides an algorithm to decide this property in the linear case, relating these topics.

1.2 Tree automata

In the same way that finite automata recognize regular languages of words, tree automata recognize languages of trees called regular tree languages also. We recall some definitions and results, and we refer the reader to [GS84] for more details.

Definition 1 *A bottom-up tree automaton $\mathcal{A}$ is a quadruple (F, Q, Q_{Final}, R) where F is the signature, Q is a finite set of states, $Q_{Final} \subseteq Q$ is a set of final states, and*

R is a set of transition rules of the form $f(q_1, \ldots q_n) \rightarrow q_{n+1}$ with $arity(f) = n$ and $q_i \in Q$.

The transition relation $\rightarrow$ is defined by $t \rightarrow t'$ iff $t = C[f(q_1(t_1), \ldots, q_n(t_n))]$ and $t' = C[q_{n+1}(f(t_1, \ldots, t_n))]$ (the q_i s are states of $\mathcal{A}$). The language accepted by $\mathcal{A}$ is the set of ground terms t such that $t \xrightarrow{*} q(t)$ where $q \in Q_{Final}$ and $\xrightarrow{*}$ is the reflexive transitive closure of $\rightarrow$. A set of ground terms is a regular tree language iff it is accepted by a tree automaton.

Example If $F = \{0, s, f\}$, the automaton $\mathcal{A} = (F, \{q_0, q_s, q_F\}, \{q_F\}, R)$ with R being $\{0 \rightarrow q_0, s(q) \rightarrow q_s$ for any $q \in Q, f(q_0, q) \rightarrow q_F$ for any $q \in Q, f(q, q') \rightarrow q_s$ for any $q \in Q - \{q_0\}, q' \in Q\}$, recognizes the language composed of the ground instances of $f(0, x)$ ◀

A tree t reaches a state q iff $t \xrightarrow{*} q(t)$. An automaton is completely specified iff for each ground term t there is a state q s.t. t reaches q. It is straightforward to complete an incomplete automaton into a complete one, and we usually write incomplete automata, dropping the uninteresting error states and rules required to define a completely specified automaton. An completely specified automaton is deterministic when each tree reaches one and only one state. Tree automata enjoy good properties: the emptiness of the accepted language is decidable, there exists a determinization algorithm as well as a minimization algorithm. Moreover regular tree languages are closed under union, intersection and complement.

1.3 Flattened terms

A function symbol which is AC can be seen as having an arbitrary arity, and we can *flatten* terms by erasing all useless occurrences of AC-symbol using the simplification $f(\ldots, f(s_1, \ldots, s_m), \ldots) \rightarrow f(\ldots, s_1, \ldots, s_m, \ldots)$. When no rewriting is possible, one has a *flattened* term which is unique up to commutativity since the equivalence on terms modulo AC becomes an equivalence on flattened terms modulo C. If t is a term, $flat(t)$ is one of its flattened versions. All previous definitions on terms are still valid on flattened terms. For simplicity, we allow the notation $f(t)$ which denotes t if $f \in F_{AC}$, for example $f(0)$ is 0. In what follows, multisets i.e sets with repeated elements, are denoted by $\{\{\ldots\}\}$. We give some results which are useful in the following:

- $s =_{AC} t$ iff $flat(s) =_C flat(t)$
- $flat(t) =_C f(t_1, \ldots, t_n)$ iff $t = f(s_1, s_2)$ with $flat(s_1) =_C f(t_{i_1}, \ldots, t_{i_p})$, $flat(s_2) =_C f(t_{j_1}, \ldots, t_{j_l})$ and the multiset $\{\{t_{i_1}, \ldots, t_{i_p}, t_{j_1}, \ldots, t_{j_l}\}\}$ is equal to the multiset $\{\{t_1, \ldots, t_n\}\}$
- if $s =_{AC} t\sigma$ where $flat(t) =_C f(t_1, \ldots, t_n, x_1, \ldots, x_m)$ and $t_i \notin X$, and $flat(t_i\sigma) = v_i$, $flat(x_i\sigma) = f(u_i^1, \ldots, u_i^{n_i})$ then $s = f(s_1, s_2)$ s.t. $flat(s_1) =_C f(v_{i_1}, \ldots, v_{i_k}, \ldots, u_{j_l}^{k_l}, \ldots)$ and $flat(s_2) =_C f(v_{i'_1}, \ldots, v_{i'_{k'}}, \ldots, u_{j'_l}^{k'_l}, \ldots)$ and the multiset $\{\{v_1, \ldots, v_n\}\}$ is the union of the multisets $\{\{v_{i_1}, \ldots, v_{i_k}\}\}$ and

$\{\{v_{i'_1}, \ldots, v_{i'_{k'}}\}\}$, the multiset $\{\{u_1^1, \ldots u_1^{n_1}, \ldots, u_m^1, \ldots, u_m^{n_m}\}\}$ is the union of the multiset $\{\{\ldots, u_{j_l}^{k_l}, \ldots\}\}$ and of the multiset $\{\{\ldots, u_{j'_l}^{k'_l} \ldots\}\}$.

2 Tree automata solution for linear AC-problems

Tree automata will provide a nice and short solution to linear problems modulo AC. Throughout this section we assume that $F = F_{AC} \cup F_{NAC}$ where all symbols in F_{AC} are AC, and all symbols in F_{NAC} are free.

2.1 Tree automata solve linear AC-problems

The key result on which our decidability result relies is:

Proposition 1 *Let t be a linear term, then the set of ground AC-instances of t is a regular tree language.*

Proof. We show that the set of ground AC-instances of t satisfies a least fixed-point equation defining regular languages. At the same time, we sketch the construction of an automaton accepting this language. The proof is by structural induction on t.

- $t \in X$ or t is a constant: the proof is obvious.
- $t = f(t_1, \ldots, t_n)$ with $f \in F_{NAC}$: let L_i be the language of the ground AC-instances of t_i. By induction hypothesis, L_i is a regular language, let $\mathcal{A}_i = (Q^i, Q_F^i, R_i)$ be an automaton accepting L_i. Let L be the set of ground AC-instances of t, then $L = f(L_1, \ldots, L_n)$ which proves the result. An automaton accepting L is $\mathcal{A} = (Q, Q_F, R)$ where $Q = \cup Q_i \cup \{q_F\}$, $Q = \{q_F\}$, $R = \cup R_i \cup \{f(q_1, \ldots, q_n) \rightarrow q_F \ where \ q_i \in Q_F^i\}$.
- $t = f(\ldots)$ with $f \in F_{AC}$: let $flat(t) = f(t_1, \ldots, t_n)$. Let L be the set of ground AC-instances of t, for each $I = \{i_1, \ldots, i_k\} \subseteq \{1 \ldots n\}$ let L_I be the set of ground AC-instances of a term t_I s.t. $flat(t_I) = f(t_{i_1}, \ldots, t_{i_k})$. By induction hypothesis, this set is regular whenever $I \subset \{1 \ldots n\}$, and let $\mathcal{A}_I = (F, Q^I, Q_F^I, R_I)$ be an automaton recognizing this set. Then, according to the results on flattened terms recalled in section 1.3 and using the fact that t is linear, we can write the equation:

$$L = \sum_{\substack{I \cup J = \{1 \ldots n\} \\ I, J \neq \emptyset \\ i \in I \cap J \Rightarrow t_i \in X}} f(L_I, L_J)^1$$

Together with the equations defining the L_I's for I a strict subset of $\{1 \ldots n\}$ (by induction hypothesis), one gets a system such that its least fixed-point solution is a t-uple of regular languages, which proves the result.
Moreover, an automaton accepting L is $\mathcal{A} = (F, Q, Q_F, R)$ where

[1] when I or J is $\{1 \ldots n\}$ replace L_I or L_J by L in the above equation

- $Q = \cup_I Q_I \cup \{q_F\}$
- $Q_F = \{q_F\}$
- $R = \cup_I R_I \cup \{f(q_I, q_J) \rightarrow q_F \text{ for all } q_i \in Q_F^I, q_j \in Q_F^J s.t.\ I \cup J = \{1 \ldots n\}, i \in I \cap J \Rightarrow t_i \in X\}$

□

Example : Let $t = f(0, x)$ where $F_{AC} = \{f\}$ and $F_{NAC} = \{0, s\}$. Let L be the language of the ground AC-instances of t, let L_0 the the language of the ground AC-instances of 0, let L_x be the language of the ground AC-instances of x, the system of equations defining L is: $L_0 = \{0\}$
$L_x = L_0 + s(L_x) + f(L_x, L_x)$
$L = f(L_0, L_x) + f(L_x, L_0) + f(L, L_x) + f(L_x, L)$
and an automaton recognizing this language is $\mathcal{A} = (\{q_x, q_0, q_L\}, \{q_L\}, R)$ with R consisting of: $0 \rightarrow q_0$, (the rule of an automaton recognizing L_0), $0 \rightarrow q_x$, $s(q_x) \rightarrow q_x$, $f(q_x, q_x) \rightarrow q_x$ (the rules of an automaton recognizing the instances of x) and $f(q_0, q_x) \rightarrow q_L, f(q_x, q_0) \rightarrow q_L, f(q_x, q_L) \rightarrow q_L, f(q_L, q_x) \rightarrow q_L$. This automaton is not completely specified nor deterministic but it can be completed and determinized ◄

This property gives the foundations for the decidability result we are searching for:

Theorem 1 *The complement problem $t \neq_{AC} t_1 \wedge \ldots \wedge t \neq_{AC} t_n$ where the t_i's are linear terms is decidable.*

Proof. Let $t \neq_{AC} t_1 \wedge \ldots \wedge t \neq_{AC} t_n$ be a complement problem s.t the t_i's are linear (but t may be non-linear). Let L be the set of the ground AC-instances of the t_i's, then L is a regular language since the t_i's are linear. Let L_C be the complement of L and $\mathcal{A}_C$ be an automaton recognizing L_C. If there is a ground instance $t\theta$ of t which is a solution of the complement problem, then this instance is accepted by $\mathcal{A}_C$. For each variable x_i of t, $x_i\theta$ is some ground term and $x_i\theta \xrightarrow{*} q_i(x_i\theta)$ for some state q_i of $\mathcal{A}_C$. Therefore there is a solution of the complement problem iff there is some assignment of states to the variables of t s.t. $t_{[x_1 \leftarrow q_1(x_1), \ldots x_n \leftarrow q_n(x_n)]} \xrightarrow{*} q_f(t)$ where q_f is a final state of $\mathcal{A}_C$ and where $t_{[x_1 \leftarrow q_1(x_1), \ldots, x_n \leftarrow q_n(x_n)]}$ denotes t where each occurrence of x_i is replaced by $q_i(x_i)$. If there is no such assignment, then there is no solution to the complement problem. Since there is a finite number of possible assignments, the complement problem modulo AC is decidable.
□

2.2 What's the trouble with non-linearity

Before giving some extensions of the previous result, we show which problem arises when we consider non-linear terms. Even in the empty theory, non-linearity causes troubles, and tree automata for non-linear terms need equality tests, which yields the undecidability of decision of emptiness of the accepted language. Moreover specific

problems arise when dealing with the AC-theory, as shown in the following example. Let F_{AC} consist of one symbol f , and let F_{NAC} consist of two constants 0 and 1 and let t be $f(x,x)$. Then the ground AC-instances of $f(x,x)$ are difficult to recognize since some of these instances are trees such that the first son and its brother are not equal, even modulo AC. For instance $f(0,f(1,f(1,0)))$ and $f(1,f(0,f(1,0)))$ are ground AC-instances of $f(x,x)$. To know that a ground term (with root f) is an instance of $f(x,x)$, one must count $0's$ and $1's$ and there is no way to conclude before all $0's$ and $1's$ have been counted: the ground term is an AC-instance of $f(x,x)$ if there is an even number of $0's$ and an even number of $1's$. Usual tree automata cannot cope with this problem. We shall present new kinds of tree automata more powerful than classical ones, which help solve some non-linear AC-complement cases.

3 Extending the results

Our solution to AC-complement can be extended in several ways. The first one deals with the inductive reducibility modulo AC, the second one deals with some related theories, and the third one is to weaken the linearity requirement.

3.1 To the inductive reducibility modulo AC

A term is inductively reducible, iff all its ground instances are reducible. More precisely, given a rewrite system R and a set of AC-axioms, a term t is reducible iff $t =_{AC} t'$ and there is a subterm $t'_{|p}$ of t' at position p such that $t'_{|p}$ is equal to $l\theta$, an instance of a left-hand side of a rule $l \rightarrow r$. Therefore a term is reducible if it is equal modulo AC to a term $C[l\theta]$ where C is a context, l is a left-hand side of a rule and θ is a ground substitution. We now prove that the set of terms equal modulo AC to $C[l\theta]$ where θ ranges over the set of ground substitutions, is a regular tree language when l is linear.

Proposition 2 *Let t be a linear term, then the set L of terms $Red(t) = \{s \text{ s.t. } s =_{AC} C[t\theta]$ for some context C and some ground substitution $\theta\}$ is a regular tree language.*

Proof. The proof is similar to the proof of proposition 1.

1. t is a constant or a variable: obvious.

2. $t = f(t_1,\ldots,t_n)$ where $F \in F_{NAC}$. A ground term is reducible by t if it contains a ground AC-instance of t. Let L_I be the languages consisting of the ground AC-instances of the t_i 's and let L be the language of the ground terms reducible by t, then L satisfies the equation $L = f(L_1,\ldots,L_n) + \sum_{g \in F} g(L \text{ or } L_\emptyset,\ldots,L \text{ or } L_\emptyset)$ where $L_\emptyset$ is the set of all ground terms and where at least one argument of the g 's occurring in the sum is L.

3. t is s.t. $flat(t) =_C f(t_1,\ldots,t_n)$. Let L_I be the set of ground AC-instances of a term t_I s.t. $flat(t_I) = f(t_{i_1},\ldots,t_{i_k})$ and $I = \{i_1 \ldots i_k\}$ and let L be the set of ground terms reducible by t. By induction hypothesis, this set is regular

whenever $I \subset \{1 \ldots n\}$. Then L satisfies the equation:

$$L = \sum_{\substack{I \cup J = \{1 \ldots n\} \\ I, J \subsetneq \{1 \ldots n\}}} f(L_I, L_J) + \sum_{g \in F} g(L \text{ or } L_\emptyset, \ldots, L \text{ or } L_\emptyset)$$

where at least one argument of the g 's occurring in the sum is L.

□

From this result, we get the decision of the inductive-reducibility modulo AC.

Theorem 2 *The inductive reducibility modulo AC of a term t for a left-linear system R is decidable.*

Proof. Let $l_1, \ldots, l_n$ be the set of left-hand sides of R, and let $\mathcal{A}$ be a deterministic automaton recognizing the union of the $Red(l_i)$'s. The term t is inductively reducible iff for each ground θ, $t\theta$ belongs to this set. We proceed as in the proof of the complement modulo AC: to each variable x_i of t assign some state q_i, and compute the state q such that $t_{[x_1 \leftarrow q_1(x_1), \ldots, x_n \leftarrow q_n(x_n)]} \xrightarrow{*} q(t)$. If there is some assignment such that the state q is not a final state of $\mathcal{A}$, then t is not inductively reducible, otherwise t is inductively reducible. Since there is a finite number of possible assignments, the inductively reducibility property is decidable.
□

3.2 To some other theories

The previous results can be easily extended to other theories. The first one is the AC1 theory, i.e AC with unity: for each AC-symbol f, there exists a constant e such that $f(x, e) = x$. To handle this identity element, we add the terms $f(L_e, L)$ and $f(L, L_e)$ to the equations defining the ground AC-instances of a term $t = f(\ldots)$ where $L_e = \{e\} + f(L_e, L_e)$. From the automaton viewpoint, it amounts to add rules $f(q, q_e) \rightarrow q$, $f(q_e, q) \rightarrow q$ where q is a final state, and $e \rightarrow q_e$, $f(q_e, q_e) \rightarrow q_e$.

The second one is the associativity theory. In this case the flattened version of a term is unique since the commutativity axioms does not hold. The proof that the A-instances of a linear term is a regular language works as in the AC case for the first two cases, and in the proof of the third case, sets are replaced by lists: $\{1 \ldots n\}$ becomes $[1 \ldots n]$, subsets I and J becomes sublists and the union is replaced by concatenation.

Theorem 3 *The complement problem $t \neq_{AC1} t_1 \wedge \ldots \wedge t \neq_{AC1} t_n$ (resp. $\neq_A$) where the t_i 's are linear terms is decidable. The inductive reducibility modulo AC1 (resp. modulo A) of a term t for a left-linear term rewrite system is decidable.*

3.3 To some non-linear cases

Tree automata with syntactical equality tests between brothers are introduced in [BT92] where it is shown that this classs is closed under boolean operations and that

the emptiness of the accepted language is decidable. We extend this class by allowing equality tests modulo AC in order to get the decidability of the AC-complement problem for *strictly restricted* non-linear terms, as defined in the next definition.

Definition 2 *The non-linearity of a term is strictly restricted iff for each non-linear variable x, there exists a position p such that all the occurrences of x occur at positions $p.i$ with i an integer, and the symbol at position p is not AC.*

For example, if $F_{AC} = \{f\}$ and $F_{NAC} = \{0, g\}$ then $f(g(x,x),0)$ is strictly restricted but $f(x,x)$ and $f(x,g(0,x))$ are not. The approach of Bogaert and Tison generalizes smoothly to the AC case, except that the decision of emptiness requires that equivalent terms reach the same states, which is true in our applications. The reader is referred to the original work of [BT92] for details since the algorithms are identical except that $=$ is replaced by $=_{AC}$. First we define our new class of tree automata.

Definition 3 *An automaton with equality tests between brothers $\mathcal{A}$ is a quadruple (F, Q, Q_F, R) where Q is a finite set of states, $Q_F \subseteq Q$ is a set of final states and R is a set of rules $< \varphi >: f(q_1, \ldots, q_n) \rightarrow q_{n+1}$ with $\varphi \in Form_n$ where $Form_n$ is inductively defined by:*

- $\#i =_{AC} \#j \in Form_n$ *(which means that the i^{th} son is equal to the j^{th} son modulo AC),* $\top \in Form_n$ *(meaning that no condition is required)*
- *if $\varphi \in Form_n$ and $\psi \in Form_n$ then $\neg\varphi \in Form_n, \psi \vee \varphi \in Form_n, \psi \wedge \varphi \in Form_n$.*

Moreover if $f \in F_{AC}$ and $< \varphi >: f(q_1, q_2) \rightarrow q_3 \in R$ we demand that φ is $\top$ (i.e there is no condition for rules with AC symbols). *The transition function is defined by $t \rightarrow t'$ iff $t = C[g(q_1(t_1), \ldots, q_n(t_n))]$, $t' = C[q_{n+1}(f(t_1, \ldots, t_n))]$ and $(t_1, \ldots, t_n)$ satisfies φ. The reflexive transitive closure of $\rightarrow$ is denoted by $\xrightarrow{*}$. The accepted language $\mathcal{L}(\mathcal{A})$ is the set of trees t such that $t \xrightarrow{*} q(t)$ with $q \in Q_F$.*

One should notice that equality tests are allowed under non-AC symbols only. An incompletely specified tree automaton with equality tests can be easily extended into a completely specified one. Now we sketch the determinization process which is similar to the determinization process for tree automata with syntactical equality tests.

1. Separate the conditions such that the conditions becomes mutually exclusive (for example use the subset $\bigwedge_{i,j \in I} \#i =_{AC} \#j \bigwedge_{i,j \in [1\ldots,n]-I} \#i \neq_{AC} \#j$ of $Form_n$)
2. Use a usual determinization algorithm to compute the equivalent deterministic automaton (a state of the deterministic automaton is a set of states of the non-deterministic one)

Remark. : let t be a term and let $q_1, \ldots, q_n$ be the states such that $t \xrightarrow{*} q_i(t)$ in the non-deterministic automaton then $t \xrightarrow{*} \{q_1, \ldots, q_n\}(t)$ in the deterministic one.

A direct consequence of the determinization process is that the class of accepted language is closed under complement (exchange final and non-final states). Moreover it is closed under union (straightforward) hence under intersection. What remains to give is a way to decide the emptiness of the language accepted by a automaton with equality tests. It works as in [BT92], *provided that equivalent terms reach the sames states, i.e* $t =_{AC} t'$ *and* $t \xrightarrow{*} q(t)$ *implies that* $t' \xrightarrow{*} q(t')$. Fortunately, this property is preserved under determinization (see the previous remark) and union, intersection and complement. From now we suppose that this property is satisfied by the automata that we consider. We need first some definitions to prove a technical lemma. Let $< \varphi >: g(q_1, \ldots, q_n) \rightarrow q_{n+1}$ be a rule r, and suppose that α_i trees of distinct equivalence classes modulo AC reach the state q_i for $1 \leq i \leq n$. Let $N_r(\alpha_1 \ldots, \alpha_n)$ be the number of equivalence classes modulo AC such that, for each class, at least one tree of this equivalence class reaches q_{n+1} using this rule at the top. Then the following implications hold (*max-arity* denotes the maximal arity of function symbols):

Proposition 3

- $\exists i : \alpha_i \geq$ max-arity $\Rightarrow N_r(\alpha_1, \ldots, \alpha_n) = 0$ *or* $N_r(\alpha_1, \ldots, \alpha_n) \geq$ max-arity
- $\exists \alpha_i : N_r(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_n) \geq$ max-arity $\Rightarrow N_r(\alpha_1, \ldots,$max-arity$, \ldots, \alpha_n) \geq$ max-arity.

Proof. The same as in [BT92], replacing $=$ by $=_{AC}$ and using the condition on equivalent terms. □

The algorithm for deciding the emptiness of the language accepted by an automaton with equality tests modulo AC between brothers is a consequence of the above result. In the algorithm, $N_{AC}(L)$ denotes the number of distinct equivalence classes of a finite set of ground terms L.

For each state q **do** set $\mathcal{L}_q^0 = \emptyset$.
i=1.
Repeat
 For each state q **do** $\mathcal{L}_q^i = \emptyset$
 For each rule r of the form $< \varphi >: h(q_1, \ldots, q_n) \rightarrow q$ **do**
 if $N_{AC}(\mathcal{L}_q^{i-1}) \geq$ max-arity **then** $\mathcal{L}_q^i = \mathcal{L}_q^{i-1}$
 else $\mathcal{L}_q^i = \mathcal{L}_q^{i-1} \cup \{h(t_1, \ldots, t_n)$ *satisfying* φ *where* $t_j \in \mathcal{L}_{q_j}^{i-1}\}$
 i=i+1
until $\exists q \in Q_F$ s.t. $\mathcal{L}_q^i \neq \emptyset$ or $\forall q, \mathcal{L}_q^i = \mathcal{L}_q^{i-1}$
if $\exists q \in Q_F$ s.t. $\mathcal{L}_q^i \neq \emptyset$ **then return** *not empty*
else return *empty*

Proposition 4 *The emptiness decision algorithm terminates and is correct.*

Proof. The correctness and termination of the algorithm rely on the previous proposition and on the fact that equivalence classes modulo AC are finite. The algorithm constructs an increasing sequence of languages. At each step a new term is added in some $\mathcal{L}_i$ otherwise the algorithm stops. Since the AC-equivalence class of a term is finite, either the bound max-arity is reached or no new term is added to any $\mathcal{L}_i$ which proves that the algorithm terminates. The previous proposition proves that it is sufficient to have *max-arity* distinct equivalence classes in $\mathcal{L}_i$ to decide if a rule applies, therefore the algorithm is correct.
□

Now we prove that the ground AC-instances of a strictly restricted term are recognized by a tree automaton with equality tests which satisfies the condition on equivalent terms.

Proposition 5 *Let t be a strictly restricted term, then the set of ground AC-instances of t is accepted by a tree automaton with equality tests.*

Proof. The proof is as in the linear case with one difference: if $t = f(\ldots)$ with $f \in F_{NAC}$, for each non-linear variable x occurring as the $i_1^{th}, i_2^{th}, \ldots, i_n^{th}$ sons of t, add the condition $\#i_1 =_{AC} \#i_2 =_{AC} \ldots =_{AC} \#i_n$ to the related rule of the automaton. □

The theorem on the decidability of complement problem and inductive reducibility is an immediate consequence of the previous results.

Theorem 4 *The complement problem $t \neq_{AC} t_1 \wedge \ldots \wedge t \neq_{AC} t_n$ (resp. $\neq_A$) where the t_i 's are strictly restricted, is decidable. The inductive reducibility modulo AC (resp. modulo A) of a term t for a term rewrite system with strictly restricted left-hand sides is decidable.*

Proof. The proof is the same as for the linear case. □

The next extension that we describe will allow occurrences of non-linear variables under AC-symbols when they occur under the same node. First, we shall introduce a new class of tree automata, *conditional tree automata* and we shall study its properties. After that, we show how this class can be used to solve AC-complement problems and to decide the inductive reducibility property in this restricted non-linear case.

4 Conditional tree automata

4.1 Normalized Flattened terms

To deal with non-linearity, we shall work on flattened terms where all subterms equal modulo AC have been grouped and counted together, like in $f(2.0, 3.s(0))$[2]. This transformation will allow to deal with the ground AC-instances of terms where the

[2]grouping identical terms is essential in our approach but prevent us from using the algebraic approach which works nicely in the linear case

non-linear variables occurs under the same AC-symbol once the flattening process is done i.e terms such as $f(f(x,0),f(0,x))$ (since $flat(t) = f(x,0,0,x)$). First we define this new kind of flattened terms, then we define the related class of tree automata.

Definition 4 *A flattened term with counter is*

- *a variable or a constant,*
- *a term* $f(t_1,\ldots,t_n)$ *where* $f \in F_{NAC}$ *and* t_i *are flattened terms with counter,*
- *a term* $f(n_1.t_1,\ldots,n_p.t_p)$ *where* $f \in F_{AC}$ *and* t_i *are flattened terms with counter.*

For example a flattened term $f(t_1,\ldots,t_n)$ is a flattened term with counter $f(1.t_1,\ldots,1.t_n)$. We extend the relation $=_C$ on flattened term with counter as follows:

Definition 5 *The relation* $=_C$ *is defined by:*

- $s =_C t$ *if* s *and* t *are syntactically equal,*
- $f(s_1,\ldots,s_n) =_C f(t_1,\ldots,t_n)$ *if* $f \in F_{NAC}$ *and* $s_i =_C t_i$ *for* $i = 1\ldots n$,
- $f(n_1.s_1,\ldots,n_p.s_p) =_C f(m_1.t_1,\ldots,m_k.t_k)$ *if* $f \in F_{AC}$ *and there is a bijection between* $\{s_1,\ldots,s_p\}$ *and* $\{t_1,\ldots,t_k\}$ *s.t. if* s_i *is mapped on* t_j *then* $s_i =_C t_j$ *and* $n_i = m_j$.

For example, $g(0,f(2.0,s(0))) =_C g(0,f(s(0),2.0))$ if $f \in F_{AC}$ and $g \in F_{NAC}$. To group identical subterms as much as possible, we use the reduction relation $\rightsquigarrow$:

Definition 6 *The reduction relation* $\rightsquigarrow$ *is defined by:*

- *If* $f \in F_{NAC}$ *and* $t_i \rightsquigarrow t_j$ *then* $f(\ldots,t_i,\ldots) \rightsquigarrow f(\ldots,t_j,\ldots)$.
- *If* $t_i =_C t_j$ *and* $f \in F_{AC}$ *then* $f(\ldots,n_i.t_i,\ldots,n_j.t_j,\ldots) \rightsquigarrow f(\ldots,(n_i+n_j).t_i,\ldots)$

This reduction relation terminates and is locally confluent up to commutativity, yielding a normalized (flattened) term unique up to commutativity. A normalized form of a term t *is a normalized form of* $flat(t)$.

For example $f(f(x,s(0)),f(s(0),x))$ has two normalized forms $f(2.x,2.s(0))$ and $f(2.s(0),2.x)$ which are identical up to permutations of the arguments of f. The usual notions on terms extend to normalized forms in a straightforward way. A useful one is that of the equivalent class of a normalized term t which is the set of normalized terms equal to t modulo commutativity. The next step is to design tree automata, called *conditional tree automata*, accepting sets of normalized terms.

4.2 Conditional tree automata

Since normalized terms have numerical counters, we shall use formulae to deal with these counters. These formulae define conditions to check before applying a rule. To get a class of automata which has good properties, we shall require that these formulae belong to a decidable theory. Presburger's arithmetic is well suited to our purpose (the AC-complement problem) but the results on conditional tree automata still hold if Presburger's arithmetic is replaced by another decidable theory.

Definition 7 *A conditional tree automata $\mathcal{A}$ is a quadruple (F, Q, Q_{Final}, R) where Q is a finite set of states, $Q_{Final} \subseteq Q$ is a set of final states, and R is a set of transition rules.* Sorts *are symbols of the form f or $\neq f$ where $f \in F_{AC}$, and each state has one or several sorts but cannot have both sorts f and $\neq f$. The transition rules are of the form:*

- $f(q_1, \ldots, q_p) \rightarrow q$ *if* $f \in F_{NAC}$ *with* $arity(f) = p$
- $- < \varphi(N) >: f(N.q_1) \rightarrow q_f$ *where* q_1 *has sort* $\neq f$
  $- < \varphi(N) >: f(N.q_1, q_2) \rightarrow q_f$

  if $f \in F_{AC}$,
  where N is an integer variable and φ a formula of Presburger's arithmetic which has a unique free variable N, q_1 has sort $\neq f$ and q_f has sort f.

The transition relation $\rightarrow$ is defined on normalized terms by $t \rightarrow t'$ iff:

- *either* $t = C[f(q_1(t_1), \ldots, q_p(t_p))]$ *with* $f \in F_{NAC}$ *and* $t' = C[q(f(t_1, \ldots, t_p))]$ *and* $f(q_1, \ldots, q_p) \rightarrow q \in R$
- *or* $(f \in F_{AC})$
  1. $t = C[f(n_1.q_1(t_1))]$ *with* $n_1 > 1$ *and* $t' = C[q_f f(n_1.t_1)]$ *if there is a rule* $< \varphi(N_1) >: f(N_1.q_1) \rightarrow q_f$ *such that* $\varphi(n_1)$ *is true.*
  2. $t = C[f(n_1.q_1(t_1), q_2(t_2))]$ *and* $t' = C[q_f f(n_1.t_1, t_2)]$, q_2 *has sort* $\neq f$ *and there is a rule* $< \varphi(N_1) >: f(N_1.q_1, q_2) \rightarrow q_f$ *such that* $\varphi(n_1)$ *is true.*
  3. $t = C[f(n_1.q_1(t_1), \ldots, n_p.q_p(t_p))]$ *and* $t' = C[q_f f(n_1.t_1, \ldots, n_p.t_p)]$, *where* $f(n_2.t_2, \ldots, n_p.t_p) \xrightarrow{*} q'_f(f(n_2.t_2, \ldots, n_p.t_p))$, *and there is a rule* $< \varphi(N_1) >: f(N_1.q_1, q'_f) \rightarrow q_f$ *such that* $\varphi(n_1)$ *is true.*

where $\xrightarrow{*}$ is the reflexive transitive closure of $\rightarrow$. The language $\mathcal{L}(\mathcal{A})$ accepted by $\mathcal{A}$ is the set of ground normalized terms such that $t \xrightarrow{*} q(t)$ where $q \in Q_{Final}$.

Remark. q has sort f and $t \xrightarrow{*} q(t)$ implies that the root of t is f, and q has sort $\neq f$ and $t \xrightarrow{*} q(t)$ implies that the root of t is not f.
Example Let $F_{NAC} = \{0\}$ and $F_{AC} = \{f\}$ and $\mathcal{A} = (F, \{q_0\}, \{q_S\}, R)$ with q_0 not of sort f and q_S of sort f, where R is $0 \rightarrow q_0$, $< \exists k : N = 2k >; f(N.q_0) \rightarrow q_S$. The normalized term $f(2.0)$ is accepted by $\mathcal{A}$ since $f(2.0) \xrightarrow{*} q_S(f(2.0))$. Actually, $\mathcal{L}(\mathcal{A})$ is the set of normalized ground AC-instances of $f(x, x)$ ◀

An automaton is *completely specified* if for each normalized ground term t there is a state q such that $t \xrightarrow{*} q(t)$. A completely specified automaton can be easily derived from an incomplete one: for each symbol f add a new state q_e^f which denotes an error state of sort f for normalized terms with root f, and new rules to deal with missing cases (with a condition equivalent to $\top$).

4.3 Determinization of conditional tree automata

4.3.1 Separation of conditions

The first step in the determinization of conditional tree automata is to get rid of possible overlaps of conditions, i.e to ensure that either the conditions of any two rules cannot not be satisfied simultaneously or are equivalent. We shall use the following proposition:

Proposition 6 *Let $\varphi_1(N), \ldots, \varphi_n(N)$ be formulae of Presburger's arithmetic with free variable N, then there exist $\psi_1(N), \ldots, \psi_m(N)$ such that*

- *$\psi_i(N) \wedge \psi_j(N)$ is unsatisfiable.*
- *$\varphi_i(N) \Leftrightarrow \psi_{j_1}(N) \vee \ldots \vee \psi_{j_i}(N)$*

Proof. Define ψ_j as a suitable boolean combination (using $\neg, \vee, \wedge$) of the φ_i 's. □

The result on the separation on conditions follows: Let $\varphi_1(N), \ldots, \varphi_n(N)$ be the conditions appearing in the rules of a conditional tree automaton, then compute the related $\psi_j's$ and replace a rule $< \varphi_i(N) >: f(\ldots) \rightarrow \ldots$ by the set of rules $< \psi_{j_1}(N) >: f(\ldots) \rightarrow \ldots, \ldots, < \psi_{j_i}(N) >: f(\ldots) \rightarrow \ldots$. After this transformation either two rules have the same condition or their conditions cannot be satisfied simultaneously. The conditions are said to be *separated.* Moreover we shall assume that all rules with unsatisfiable conditions are discarded, as for the rules $< \varphi(N) > f(N.q) \rightarrow q'$ where the n satisfying $\varphi(N)$ are less than 2.

4.3.2 The determinization algorithm

Now, we show how to construct a deterministic automaton $\mathcal{A}_D = (F, \mathcal{Q}_D, \mathcal{Q}_{DF}, \mathcal{R}_D)$ from a completely specified non-deterministic automaton with separated conditions $\mathcal{A} = (F, \mathcal{Q}, \mathcal{Q}_F, \mathcal{R})$. Moreover if $\{q_1, \ldots, q_p\}$ is the set of states such that $t \xrightarrow{*} q_i(t)$ with the non-deterministic automaton, then $t \xrightarrow{*} \{q_1, \ldots, q_p\}(t)$ in the deterministic one.

A state Q of the deterministic automaton $\mathcal{A}_D$ is a set $\{q_1, \ldots, q_p\}$ of states of the non-deterministic one $\mathcal{A}$, and the rules are:

A state Q of $\mathcal{Q}_D$ is a final state iff it contains a final state of $\mathcal{A}$.

- if $f \in F_{NAC}, f(Q_1, \ldots, Q_p) \rightarrow Q$ where Q is the set of states q such there exist $q_1 \in Q_1, \ldots, q_p \in Q_p$ and a rule $f(q_1, \ldots, q_p) \rightarrow q$
- if $f \in F_{AC}$, for each $\psi_i(N)$ condition of the non-deterministic automaton,

- $< \psi_i(N) >: f(N.Q) \rightarrow Q_f$ where each state $q \in Q$ has sort $\neq f$, and where Q_f is the set of states q_f such there exists a state $q \in Q$ and a rule $< \psi_i(N) >: f(N.q) \rightarrow q_f$.
- $< \psi_i(N) >: f(N.Q, Q') \rightarrow Q_f$ where each state $q \in Q$ and each $q' \in Q'$ do not have sort f, and where Q_f is the set of states q_f such there exist a state $q \in Q$, a state $q' \in Q'$ and a rule $< \psi_i(N) >: f(N.q, q') \rightarrow q_f$.
- $< \psi_i(N) >: f(N.Q, Q'_f) \rightarrow Q_f$ where each state $q \in Q$ has sort $\neq f$ and where Q_f is the set of states q_f such there exist a state $q \in Q$ a state $q'_f \in Q'_f$ and a rule $< \psi_i(N) >: f(N.q, q'_f) \rightarrow q_f$.

Proposition 7 *$t \xrightarrow{*} q_i(t)$ with the non-deterministic automaton $\mathcal{A}$ for $i = 1, \ldots, p$ iff $t \xrightarrow{*} \{q_1, \ldots, q_p\}(t)$ with the deterministic automaton $\mathcal{A}_D$.*

Proof. By induction on t. □

4.4 Closure under boolean operations

As for regular tree languages, the class of languages recognized by conditional tree automata is closed under union, complement and intersection. To get an automaton recognizing the union of two languages $\mathcal{L}(\mathcal{A}_1)$ and $\mathcal{L}(\mathcal{A}_2)$ take the union of $\mathcal{A}_1$ and $\mathcal{A}_2$. To get an automaton recognizing the complement of $\mathcal{L}$, exchange final and non-final states in $\mathcal{A}$ where $\mathcal{A}$ is a completely specified deterministic automaton recognizing $\mathcal{L}$. The result on intersection is a consequence of the two previous ones.

4.5 Decision of emptiness

The main property of conditional automata is that the emptiness of the accepted language is decidable, **provided that if $t \xrightarrow{*} q(t)$ then $t' \xrightarrow{*} q(t')$ for each t' such that $t' =_C t$.** Let $\mathcal{A}$ be a deterministic completely specified conditional automaton which satisfies this condition, let $|R|$ be the cardinal of the set of rules R, and let M be *max(maximal arity of F,$|R| + 1$,3)*. In the following, we show that to distinguish between accessible and unaccessible states, it is sufficient to count up to some bound, how many different equivalence classes reach each state (we say that an equivalence class $\tilde{t}$ of normalized terms reach the state q if there is some $t \in \tilde{t}$ such that $t \xrightarrow{*} q(t)$).

First, we deal with rules of the form $f(q_1, \ldots, q_p) \rightarrow q$ with $f \in F_{NAC}$, $< \varphi(N) >: f(N.q) \rightarrow q_f$, $< \varphi(N) >: f(N.q, q') \rightarrow q_f$ (q' has sort $\neq f$).

Definition 8 *Let r be a rule of $\mathcal{A}$, let α_i for $i = 1, \ldots, n$ be integers, then we define $N_r(\alpha_1, \ldots, \alpha_n)$ as:*

- *$N_r(\alpha_1, \ldots, \alpha_p)$ is the number of equivalence classes $\tilde{t}$ such that there exists $t \in \tilde{t}$ reaching q using r at the top if r is $f(q_1, \ldots, q_p) \rightarrow q$ with $f \in F_{NAC}$ and if α_i equivalence classes reach q_i for $i = 1, \ldots, p$,*
- *$N_r(\alpha)$ is the number of equivalence classes $\tilde{t}$ such that there exists $t \in \tilde{t}$ reaching q_f using r at the top if r is $< \varphi(N) >: f(N.q) \rightarrow q_f$ with $f \in F_{AC}$ and if α equivalence classes reach q,*

- $N_r(\alpha_1, \alpha_2)$ *is the number of equivalence classes* $\tilde{t}$ *such that there exists* $t \in \tilde{t}$ *reaching* q_f *using* r *at the top if* r *is* $f(N.q_1, q_2) \rightarrow q_f$ *with* $f \in F_{AC}$ *and if* α_i *equivalence classes reach* q_i *for* $i = 1, 2$.

Then,

Proposition 8 *The two implications hold:*

- $\exists \alpha_i$ *such that* $\alpha_i \geq M$ *implies that* $N_r(\alpha_1, \ldots, \alpha_n) = 0$ *or* $N_r(\alpha_1, \ldots, \alpha_n) \geq M$
- $\exists i$ *such that* $N_r(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_n) \geq M$ *implies that* $N_r(\alpha_1, \ldots, \alpha_{i-1}, M, \alpha_{i+1}, \ldots, \alpha_n) \geq M$

Proof. The proof is straightforward except for rules $< \varphi(N) >: f(N.q_1, q_2) \rightarrow q_f$. Let n_1 be such that $\varphi(n_1)$ holds. In this proof and the following ones, C_n^p denotes the number of possible choices of p objects among n.

1. Proof of the first implication: suppose that $\alpha_1 \geq M$ different equivalence classes reach q_1. Then either no normalized term reach q_2 and $N_r(M, \alpha_2) = 0$ or there is at least one normalized term reaching q_2. If $q_1 \neq q_2$ then $N_r(\alpha_1, \alpha_2) \geq M$ since a normalized term reaching q_2 is not equivalent to a normalized term reaching q_1 [3]. If $q_1 = q_2$, we have $C_M^2 = M(M-1)/2 \geq M$ terms $f(n_1.s_1, s_2)$ which are pairwise not equivalent, such that $s_i \stackrel{*}{\rightarrow} q_i(s_i)$ and $f(s_1, s_2) \stackrel{*}{\rightarrow} q$ since s_1 and s_2 are not equal modulo AC.
2. Proof of the second implication: if $N_r(\alpha_1, \alpha_2) \geq M$, then either both of α_1, α_2 are less than M and $N_r(M, \alpha_2) \geq M$, or else α_1 or α_2 is greater than M. Without loss of generality, we assume $\alpha_1 > M$. If $q_1 \neq q_2$, we can construct M terms reaching q using r, and if $q_1 = q_2$, we can construct $M(M-1)/2 \geq M$ terms $f(n_1.s_1, s_1')$ with $s_1 \neq_{AC} s_1'$ reaching q using r.

□

To deal with rules $< \varphi(N) >: f(N.q, q_f') \rightarrow q_f$ where q_f' has sort f requires more work.

Definition 9 *Let* $N_{r_1.r_2,\ldots,r_m}(\alpha_1, \alpha_2, \ldots, \alpha_k)$ *be the number of equivalence classes* $\tilde{t}$ *such that at least one term* $t = f(n_m.s_m, \ldots, n_1.s_1) \in \tilde{t}$ *reaching* q_f *using a sequence of rules ending by* $r_1, r_2, \ldots, r_m$ *when*

- r_1 *is* $< \varphi_1(N) >: f(N.q_1, q_2) \rightarrow q_f^1$ *or* $< \varphi(N) >: f(N.q_1) \rightarrow q_f^1$,
- r_i *is* $< \varphi_i(N) >: f(N.q_i, q_f^{i-1}) \rightarrow q_f^i$ *for* $i > 1$ *and* $q_f^m = q_f$, *moreover* $r_i \neq r_j$ *if* $i \neq j$,
- $\{q_1, \ldots, q_k\}$ *is the set of the states which are not of sort* f *occurring in* $r_1, \ldots, r_m$,
- α_i *equivalence classes reach the state* q_i *for* $i = 1, \ldots, k$

[3] the hypothesis that equivalent terms reach the same state is essential here

From the definition, we see that the sequence of rules has a length bounded by $|R|$, therefore a state q cannot appear more than $|R|$ times in the sequence. Then,

Proposition 9 *The two implications hold:*

- $\exists i$ *such that* $\alpha_i \geq M$ *implies*
  that $N_{r_1 \ldots r_m}(\alpha_1, \ldots, \alpha_k) = 0$ *or* $N_{r_1 \ldots r_m}(\alpha_1, \ldots, \alpha_k) \geq M$,
- $\exists i$ *such that* $N_{r_1 \ldots r_m}(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_k) \geq M$ *implies*
  that $N_{r_1 \ldots r_m}(\alpha_1, \ldots, \alpha_{i-1}, M, \alpha_{i+1}, \ldots, \alpha_n) \geq M$

Proof.

1. Proof of the first implication: suppose that $\alpha_i \geq M$ different equivalence classes reach q_i, then either no normalized term reach q_f using $r_1 \ldots r_m$ or there is some normalized term $f(n_m.s_m, \ldots, n_1.s_1)$ reaching q_f. Suppose that q_i occurs $L \leq M$ times in the sequence $r_1, \ldots, r_m$. From $f(n_m.s_m, \ldots, n_1.s_1)$ we construct $M.(M-1)\ldots(M-L+1) > M$ normalized terms reaching q_f when M equivalence classes reach q_i by replacing the first subterm s_{i_1} reaching q_i by any element of the M equivalence classes reaching q_i, the second subterm s_{i_2} reaching q_i by any term in the $M-1$ remaining classes reaching q_i, and so on. By construction all these terms reach q_f using $r_1, \ldots, r_m$. Moreover, we can construct at least $C_M^R = M!/(M-R)!R! \geq M$ such terms which are pairwise not equivalent. It is noteworthy to see that the key property used here is that $s_i \xrightarrow{*} q_i(s_i)$ and $s_j \xrightarrow{*} q_j(s_j)$ with $q_i \neq q_j$ implies that s_i and s_j are in different classes modulo AC.
2. Proof of the second implication: if $\alpha_i < M$ the proposition is obvious, otherwise there is at least one term reaching q_f using $r_1, \ldots, r_m$ and we proceed as above to construct more than M terms reaching q_f.

□

From these results and from the fact that the equivalence class modulo AC of a normalized term is finite, we get a decision algorithm which counts the number of equivalence class reaching a state q up to the bound M. In the algorithm, $N_{eq}(\mathcal{L})$ denotes the number of distinct equivalence classes in $\mathcal{L}$.

For each state q **do** set $\mathcal{L}_q^0 = \emptyset$.
i=1.
Repeat
 For each state q **do** $\mathcal{L}_q^i = \emptyset$
  For each rule r of the form $< \varphi >: h(q_1, \ldots, q_n) \rightarrow q$ **do**
  if $N_{eq}(\mathcal{L}_q^{i-1}) \geq$ M **then** $\mathcal{L}_q^i = \mathcal{L}_q^{i-1}$
  else $\mathcal{L}_q^i = \mathcal{L}_q^{i-1} \cup \{h(t_1, \ldots, t_n)$ *satisfying* φ *where* $t_j \in \mathcal{L}_{q_j}^{i-1}\}$
 i=i+1
until $\exists q \in Q_F$ s.t. $\mathcal{L}_q^i \neq \emptyset$ or $\forall q, \mathcal{L}_q^i = \mathcal{L}_q^{i-1}$
if $\exists q \in Q_F$ s.t. $\mathcal{L}_q^i \neq \emptyset$ **then return** *not empty*
else return *empty*

5 Application to complement problems and inductive reducibility

5.1 Normalized ground AC-instances

Conditional tree automata have been introduced in order to deal with so-called *restricted* terms, as defined as follows:

Definition 10 *A term t is restricted iff all the occurrences of a non-linear variable x of $flat(t)$ occurs at position $p.i$ with i an integer, and the symbol of $flat(t)$ at position p is AC.*

Example Let $F_{AC} = \{f, h\}$ and $F_{NAC} = \{0, g\}$ then $t = g(f(f(x,y)f(x,y)), h(z,z))$ is restricted since $flat(t) = g(f(x,y,x,y), h(z,z))$, but $f(x, g(x,y))$ and $f(g(x,x), 0)$ are not ◀

Our first task is to study what are the normalized forms of the AC-ground instances of a restricted term. Since flattening and the reduction relation are canonical up to commutativity, we can apply them using any strategy. Therefore given a term t and a substitution σ, we compute a normal form of $t\sigma$ by computing the normal form $N(t)$ of t, the normal form $N(\sigma)$ of σ, and finally the normal form of the application of $N(\sigma)$ to $N(t)$. The result is one normal form of $t\sigma$ and all others are equal modulo commutativity.

Let t be such $flat(t) =_C f(\ldots)$ with $f \in F_{AC}$, and that the normal form of t is equal (up to commutativity) to $f(n_1.t_1, \ldots, n_l.t_l, m_1.x_1, \ldots, m_p.x_p)$. Let the normal form of $t\sigma$ be $f(N_1.s_1, \ldots, N_n.s_n)$ and let us see how it is constructed:

A (ground) s_i comes from one or more $t_j\sigma$, i.e $t_j\sigma \rightsquigarrow s'_i =_C s_i$, and it comes also from the x_j 's such that $x_j\sigma \rightsquigarrow f(\ldots, k_i.s'_i, \ldots)$. Therefore we have $\Sigma_{j \in I_i} n_j + \Sigma_{j \in J_i} m_j.k_j$ instances s_i where I_i is a subset -possibly empty- of $\{1, \ldots, l\}$ and J_i a subset -possibly empty- of $\{1, \ldots, p\}$. Moreover each t_i gives one and only one instance $t_i\sigma$ corresponding to one s_i, and each variable x_j must contribute to at least one s_i (but it may contribute to several ones). These conditions can be formalized by stating that the union of the sets I must be equal to $\{1, \ldots, l\}$ and that the union of the sets J must contain $\{1, \ldots, p\}$. This is summarized in the following proposition:

Proposition 10 *Let t be a term, then the set of the normalized forms of its ground AC-instances is defined by:*

- *if $t = f(t_1, \ldots, t_n)$ with $f \in F_{NAC}$ then the set of terms $f(s_1, \ldots, s_n)$ where s_i is a normalized form of a ground AC-instances of t_i.*
- *if $t = f(\ldots)$ with $f \in F_{AC}$ and the normalized form of t is (up to commutativity) $f(n_1.t_1, \ldots, n_l.t_l, m_1.x_1, \ldots, m_p.x_p)$ then the set of terms $f(N_1.s_1, \ldots, N_n.s_n)$ where $N_i = \Sigma_{j \in I_i} n_i + \Sigma_{j \in J_i} m_j.k_j$ with k_j some integer, s_i belongs to the set of normalized forms of AC-ground instances of t_j for $j \in I_i$ and x_j for $j \in J_i$, and the sets I_i and J_i satisfy the conditions that the set of the non-empty I_i 's is a partition of $\{1, \ldots, l\}$ and $\cup_{i \in \{1,\ldots,n\}} J_i$ contains $\{1, \ldots, p\}$.*

Example Let $F = \{0, s, f\}$ with f an AC-symbol, and let $t = f(x, f(x, 0))$ then a normalized form of t is $f(2.x, 0)$, and the normalized ground instances of t have the form: $f(2n_1.s(\ldots), \ldots, (2n_i + 1).0, \ldots, 2.n_n.s(\ldots))$ ◄

As previously, our solution to the restricted complement problem relies on the fact that the set of ground AC-instances of a restricted term t is accepted by a tree automaton. The main difference with the previous solution is that we consider the set of the normalized forms of ground AC-instances of t and that we use a conditional tree automaton to recognize this set. This is stated in the next proposition:

Proposition 11 *Let t be a restricted term, then the set of normalized ground AC-instances of t is a language accepted by a conditional tree automaton which satisfies the condition that equivalent terms reach the same states.*

Proof. The proof is by induction on t' a normalized form of t.

- t' is a constant or a variable. The construction is obvious.
- $t' =_C f(t_1, \ldots, t_n)$ with $f \in F_{NAC}$. By induction hypothesis, for each t_i there exists a conditional tree automaton $\mathcal{A}_i$ accepting the set of ground normalized AC-instances of t_i. A automaton accepting the set of ground normalized AC-instances of t is obtained by taking the union of these automata and adding a new state q_S of sort f and the rules $f(q_S^1, \ldots, q_S^n) \rightarrow q_S$ where q_S^i is a final state of $\mathcal{A}_i$.
- $t' =_C f(n_1.t_1, \ldots, n_l.t_l, m_1.x_1, \ldots, m_p.x_p)$ with $f \in F_{AC}$ and $t_i \notin X$. By induction hypothesis, the set of normalized forms of ground AC-instances of t_i is accepted by a conditional automaton, which satisfies the condition that equivalent terms reach the same states. Therefore for each $I \subset \{1 \ldots l\}$, there is a conditional automaton $\mathcal{A}_I = (F, Q^I, Q_F^I, R^I)$ which accepts the normalized forms of common instances of the t_i 's for $i \in I$. By convention $\mathcal{A}_I$ for I the emptyset denotes an automaton accepting the set of all normalized terms i.e the normalized forms of the instances of a variable. Using the result on normalized forms of AC-instances of a term stated before, we construct an automaton $\mathcal{A} = (F, Q, Q_F, R)$ which accepts the normalized forms of AC-instances of t in the following way:
  - the states are the set of states of the $\mathcal{A}_I$ together with new states $q_{I,J}$ where $I \subseteq \{1 \ldots l\}$ and $J \subseteq \{1 \ldots p\}$, moreover these new states have sort f,
  - the final state is $q_{\{1\ldots l\},\{1\ldots p\}}$,
  - the rules are the sets of rules of the $\mathcal{A}_I$ together with the new rules:
    * $< N = \exists k_j > 0 : N = \Sigma_{i \in I} n_i + \Sigma_{j \in J} k_j.m_j >: f(N.q) \rightarrow q_{I,J}$ for all q final state of $\mathcal{A}_I$.
    * $< N = \exists k_j > 0 : N = \Sigma_{i \in I} n_i + \Sigma_{j \in J} k_j.m_j >: f(N.q, q_{I',J'}) \rightarrow q_{I \cup I', J \cup J'}$ for all q final state of $\mathcal{A}_I$ if $I \cap I' = \emptyset$.

By construction this (non-deterministic) automaton recognizes the set of normalized ground AC-instances of t: if s is such an instance, then $s = f(N_1.s_1, \ldots, N_k.s_k)$ where $N_i = \Sigma_{j \in I} n_j + \Sigma_{j \in J} k_j.m_j$, s_i is a normalized form of a common AC-instance of the $t'_j S$ for $j \in I$, the set of the non-empty I_i 's is a partition of $\{1 \ldots l\}$ and the union of the J_i 's is $\{1 \ldots p\}$. Therefore $s \xrightarrow{*} q_{\{1\ldots l\},\{1\ldots p\}}(s)$ since $s_i \xrightarrow{*} q_{I,J}(s_i)$. Conversely, if a term s reaches the final state $q_{\{1,\ldots l\},\{1\ldots p\}}$, the definition of the rules imply that this term has the form $f(N_1.s_1, \ldots, N_p.s_p)$ where the N_i 's and the s_i 's fulfill the requirements needed for s to be a normalized form of a ground AC-instance of t. Moreover if two normalized terms of the form $f(M_1.u_1, \ldots, M_k.u_k)$ are equal modulo commutativity, they reach the same set of states $q_{I,J}$. Therefore the automaton $\mathcal{A}$ meets the requirements of the proposition and we are done.
□

Example Let $t = f(2.x, 0)$ where $F_{AC} = \{f\}$ and $F_{NAC} = \{0, s\}$. An automaton recognizing the instances of x is:
$Q = \{q, q_f\}$, $Q_F = \{q, q_f\}$ where q has sort $\neq f$ and q_f has sort f, with the rules:
$0 \rightarrow q$
$s(q \text{ or } q_f) \rightarrow q$
$< N \geq 2 >: f(N.q) \rightarrow q_f$
$< N \geq 1 >: f(N.q, q \text{ or } q_f) \rightarrow q_f$
and an automaton recognizing the instances of 0:
$Q = Q_F = \{q'_0\}$ where q_0 has sort $\neq f$ with the rule: $0 \rightarrow q'_0$
Since l and p are both equal to 1, we identify $\{1 \ldots l\}$ to $\{0\}$ and $\{1 \ldots p\}$ to $\{x\}$. The new states to add are: $q_{\emptyset,\{x\}}, q_{\{0\},\emptyset}, q_{\{0\},\{x\}}$ where $q_{\{0\},\{x\}}$ is a final state and the new rules are:

$$
\begin{aligned}
&< \exists k > 0, N = 2.k >: f(N.(q \text{ or } q_f)) \rightarrow q_{\emptyset,\{x\}} \\
&< \exists k > 0, N = 1 + 2.k >: f(N.q'_0) \rightarrow q_{\{0\},\{x\}} \\
&< \exists k > 0 : N = 2.k >: f(N.(q \text{ or } q_f), q_{\{0\},\emptyset}) \rightarrow q_{\{0\},\{x\}} \\
&< \exists k > 0 : N = 2.k >: f(N.(q \text{ or } q_f), q_{\{0\},\{x\}}) \rightarrow q_{\{0\},\{x\}} \\
&< \exists k > 0 : N = 2.k >: f(N.(q \text{ or } q_f), q_{\emptyset,\{x\}}) \rightarrow q_{\emptyset,\{x\}} \\
&< N = 1 >: f(N.q'_0, q_{\emptyset,\{x\}}) \rightarrow q_{\{0\},\{x\}}
\end{aligned}
$$

The rule $< N = 1 >: f(N.q'_0)$ has been discarded since it never applies. ◄

The decidability of restricted AC-complement problems follows:

Theorem 5 *The complement problem $t \neq_{AC} t_1 \wedge \ldots \wedge t \neq_{AC} t_n$ where t and the t_i 's are restricted, is decidable.*

Proof. If there is a solution s of the complement problem, any normalized form of this solution is not a normalized ground AC-instance of any of the $t'_i s$. Therefore the problem amounts to deciding the emptiness of the intersection of the language of normalized ground AC-instances of t and of the complement of the union of the normalized ground AC-instances of the $t'_i s$, which is decidable.
□

The same technique applies to inductive reducibility modulo AC:

Theorem 6 *The inductive reducibility modulo AC of a restricted term t for a rewriting system R with restricted left-hand sides is decidable.*

Proof. In proposition 11 we gave the construction of a tree automaton recognizing the set of normalized ground AC-instances of a term t. This construction can be slightly modified to get a tree automaton which accepts the set of normalized terms which contain a subterm which is a normalized ground AC-instance of t. The construction of the automaton is modified in the following way:

- t is a constant, a variable or $f(t_1, \ldots, t_n)$ with $t \in F_{NAC}$: proceed as in the proof of proposition 11, and add the rules $g(\ldots, q_S, \ldots) \rightarrow q_S$ for all $g \in F_{NAC}$ and similar conditional rules with conditions $< N \geq 1 >$ (q_S is the succes set of the automaton).

- $t = f(n_1.t_1, \ldots, n_n.t_n)$ with $f \in F_{AC}$. We proceed as in the previous construction but the conditions $< \exists k_j > 0 : N = \Sigma_{i \in I} n_i + \Sigma_{j \in J} k_j.m_j >$ are replaced by $< \exists k_j > 0, l_j \geq 0 : N = \Sigma_{i \in I} n_i + \Sigma_{j \in J}(l_j + k_j.m_j) >$, and add the rules $g(\ldots, q_S, \ldots) \rightarrow q_S$ for all $g \in F_{NAC}$ and similar conditional rules with conditions $< N \geq 1 >$ (q_S is the success set of the automaton i.e has the form $q_{\{1\ldots l\},\{1\ldots p\}}$).

Therefore, to decide the inductive reducibility of s modulo R is to decide the inclusion of the language of the normalized AC-ground instances in the union for $i = 1, \ldots, n$ of the languages of normalized terms reducible by l_i.
□

5.2 Allowing non-linearity on brothers under AC and non-AC symbols

For simplicity, we have separated automata with equality tests under non-AC symbols and conditional tree automata. These two classes of tree automata can be merged into a unique class.

Definition 11 *A (generalized) conditional tree automaton $\mathcal{A}$ is a quadruple (F, Q, Q_{Final}, R) where Q is a finite set of states, $Q_{Final} \subseteq Q$ is a set of final states, and R is a set of transition rules.* Sorts *are symbols of the form f or $\neq f$ where $f \in F_{AC}$, and each state has one or several sorts but cannot have both sorts f and $\neq f$. The transition rules are of the form:*

- $< \varphi >: f(q_1, \ldots, q_n) \rightarrow q$ *if $f \in F_{NAC}$ with $arity(f) = n$ where $\varphi \in Form_n$ and $Form_n$ is the smallest set of formulae containing the formulae $\#i =_C \#j$, for $1 \leq i, j \leq n$ (meaning that the i^{th} son is equal to the j^{th} son) and closed under $\vee$, $\wedge$ and negation.*

- $- < \varphi(N) >: f(N.q_1) \rightarrow q_f$ *where q_1 has sort $\neq f$*
  $- < \varphi(N) >: f(N.q_1, q_2) \rightarrow q_f$
  if $f \in F_{AC}$,
  where N is an integer variable and φ a formula of Presburger's arithmetic which has a unique free variable N, q_1 has sort $\neq f$ and q_f has sort f.

The transition relation is defined as expected, as well as other usual notions on tree automata, and the class of languages accepted by such automata is closed under the boolean operations. The proofs of closure under boolean operations and the decision of emptiness are a straightforward combination of the proofs for each subclass.

In the same way the proof that the normalized form of ground AC-instances of a terms where non-linear variables occur under the same symbol is done by combining the previous proofs, and we get the general result:

Theorem 7 *Let t and t'_is be some terms s.t that all the occurrences of a non-linear variable of these terms occurs under the same function symbol in their flattened forms (without any condition on this symbol), then the complement problem $t \neq_{AC} t_1 \ldots t \neq_{AC} t_n$ is decidable. Moreover the inductive reducibility of t for a rewrite system with left hand-sides $t_1, \ldots t_n$ is decidable.*

Remark. The same approach works for the ACI (AC and the idempotency axiom) where the normalization process contains the new rule $f(\ldots, x, \ldots, x, \ldots) \rightarrow f(\ldots, x, \ldots)$. Of course all conditions collapse to $N = 1$ in the resulting tree automata.

Conclusion

We have presented a new approach to complement problem modulo a theory by reducing it to a simpler language problem: recognize the ground E-instances of a term by a automaton which belongs to a class closed under union and complementation and where the emptiness is decidable. Using bottom-up tree automata, we are able to answer this question for linear terms in many theory including AC-like ones. Moreover we have presented a new class of tree automata, conditional tree automata, which allow to solve a subcase of the non-linear case. This class is similar to the class of automata presented at the same time by [NP93] dealing with feature trees (in fact multisets). However, proofs are differents and conditional tree automata deal has less restriction on the non-linear cases which are dealt with. A byproduct of our approach is that we can also decide the inductive reducibility modulo AC of a term t modulo a rewrite system R when the non-linearity is restricted.

References

[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.

[CL89] H. Comon and P. Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7(3 & 4):371–426, 1989. Special issue on unification. Part one.

[Col84] A. Colmerauer. Equations and inequations on finite and infinite trees. In *Proceedings of FGCS'84*, pages 85–99, November 1984.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leuven, editor, *Handbook of Theoretical Computer Science*. Elsevier Science Publishers B. V. (North-Holland), 1990.

[GS84] F. Gécseg and M. Steinby. *Tree automata*. Akadémiai Kiadó, Budapest, Hungary, 1984.

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

[LM86] Jean-Louis Lassez and K. Marriot. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3):301–318, 1986.

[LMK91] J.L. Lassez, M. Maher, and K.Marriott. Elimination of negation in term algebra. In A. Tarlecki, editor, *Proceedings 16th International Symposium on Mathematical Foundations of Computer Science, Kazimierz Dolny (Poland)*, volume 520 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 1991.

[Mah88] M. J. Maher. Complete axiomatization of the algebra of finite, rational trees and infinite trees. In *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*. COMPUTER SOCIETY PRESS, 1988.

[Mal71] A. Malc'ev. Axiomatizable classes of locally free algebra of various type. In Benjamin Franklin Wells, editor, *The Metamathematics of Algebraic Systems: Collected Papers 1936-1967*, chapter 23, pages 262–281. North Holland, 1971.

[NP93] Joachim Niehren and Andreas Podelski. Feature automata and recognizable sets of feature trees. In *Tapssoft'93*, April 1993.

[Tre92] Ralf Treinen. A new method for undecidability proofs of first order theories. *Journal of Symbolic Computation*, 14(5):437–458, 1992.

Unité de Recherche INRIA Lorraine
Technopôle de Nancy-Brabois - Campus Scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 VILLERS LES NANCY Cedex (France)

Unité de Recherche INRIA Rennes IRISA, Campus Universitaire de Beaulieu 35042 RENNES Cedex (France)
Unité de Recherche INRIA Rhône-Alpes 46, avenue Félix Viallet - 38031 GRENOBLE Cedex (France)
Unité de Recherche INRIA Rocquencourt Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)
Unité de Recherche INRIA Sophia Antipolis 2004, route des Lucioles - B.P. 93 - 06902 SOPHIA ANTIPOLIS Cedex (France)

EDITEUR
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

ISSN 0249 - 6399