Alternating Two-Way AC-Tree Automata ⋆

Kumar Neeraj Verma [1,∗]

Technische Universität München, Institut für Informatik / I2
Boltzmannstraße 3, 85748 Garching, Germany

Jean Goubault-Larrecq

LSV/CNRS UMR 8643 & INRIA Futurs projet SECSI & ENS Cachan
61, av. du président-Wilson, 94235 Cachan Cedex, France

Abstract

We explore the notion of alternating two-way tree automata modulo the theory of finitely many associative-commutative (AC) symbols. This was prompted by questions arising in cryptographic protocol verification, in particular in modeling group key agreement schemes based on Diffie-Hellman-like functions, where the emptiness question for intersections of such automata is fundamental. This also has independent interest. We show that the use of general push clauses, or of alternation, leads to undecidability, already in the case of one AC symbol, with only functions of arity zero. On the other hand, emptiness is decidable in the general case of several function symbols, including several AC symbols, provided push clauses are unconditional and intersection clauses are final. This class of automata is also shown to be closed under intersection.

Key words: associative-commutative, tree automata, two-way tree automata, alternating tree automata, branching vector addition systems with states, resolution, cryptographic protocols

---

⋆ Partially supported by the ACI VERNAM, the RNTL project EVA, the RNTL project Prouvé, and the ACI jeunes chercheurs "Sécurité informatique, protocoles cryptographiques et détection d'intrusions".

∗ Corresponding author. Tel: +49 89 28 91 81 88, Fax: +49 89 28 91 81 61.
Email addresses: verma@in.tum.de (Kumar Neeraj Verma), goubault@lsv.ens-cachan.fr (Jean Goubault-Larrecq).
URLs: http://www2.in.tum.de/~verma/ (Kumar Neeraj Verma), http://www.lsv.ens-cachan.fr/~goubault/ (Jean Goubault-Larrecq).

[1] Work done while a PhD student at LSV, CNRS UMR 8643 & INRIA Futurs projet SECSI & ENS Cachan on an MENRT grant.

1 Introduction

Automata and in particular tree automata (Gécseg and Steinby, 1997; Comon et al., 1997) are an important tool in computer science, in particular in hardware or software verification (Jouannaud, 1995). We may enrich standard tree automata with various features. One that has been considered very early is that of *two-way* tree automata, where transitions may not just build terms, but also destruct terms. Another one is *alternating* tree automata, where we may recognize not just unions but also intersections of sets of terms recognized at some states (Slutzki, 1985). A more recent one is *equational* tree automata, which do not recognize terms but terms modulo some fixed equational theory, see e.g., Lugiez (1998); Ohsaki (2001). The case of finitely many associative commutative (AC) symbols is of particular importance. The goal of this paper is to explore the combination of these features, that is, of equational, two-way, alternating automata, concentrating on the equational theory of finitely many AC symbols.

Combining two-way, possibly alternating automata with the use of equational theories is not a randomly chosen research theme. We have come to need such automata in studying automata-based cryptographic protocol verification techniques, see Monniaux (1999); Genet and Klay (2000); Goubault-Larrecq (2000); Comon et al. (2001), and extending them to sets of cryptographic primitives that obey specific algebraic laws. This is particularly useful to model protocols based on the Diffie-Hellman primitive, namely modular exponentiation (Diffie and Hellman, 1976): see Goubault-Larrecq et al. (2004) for an application of two-way AC-tree automata to the automated verification of the IKA.1 group key agreement protocol.

There are a number of questions one might ask about any family of automata, in particular the ones we are considering in this paper. The first and probably the most important is decidability of emptiness.

Then, we may inquire about closure under Boolean operations: union, intersection, complementation. As we shall see, alternating AC-automata have an undecidable emptiness problem. Removing alternation but keeping two-way transitions yields a class whose emptiness problem is decidable, as we shall show. Closure under unions is trivial.

Our import in this paper is a classification of alternating, two-way AC-tree automata relative to the question of *intersection-emptiness*: given finitely many alternating two-way AC-automata, is the intersection of their languages empty? and the related question of effectively computing intersections of two-way AC-tree automata. We shall show that the subclass of so-called *AC-standard* two-way AC-tree automata (Definition 9) can be effectively reduced to one-way AC-tree automata (Theorem 44), which are closed under intersection (Theorem 38) and whose emptiness is decidable (Lemma 17). This implies that intersection-emptiness is decidable for AC-standard two-way AC-tree automata.

While this class is enough for dealing with the verification problem we initially had in mind (see Goubault-Larrecq et al. (2004) for the application to the IKA.1 cryptographic protocol), we shall leave the case of intersection-emptiness of two-way AC-tree automata, not just the AC-standard ones, open. We conjecture that the latter is still decidable, and show a first result in this direction: intersection-emptiness of two-way AC-tree automata reduces effectively to intersection-emptiness of two-way AC^0-tree automata, i.e., to the *constant-only* subcase where the only function symbols are $+$ (associative and commutative) and finitely many constants (Proposition 63). As the reader will be quickly convinced, this is already rather technical, and requires tools from several domains, in particular from automata theory, automated deduction, and Petri nets.

Outline. The paper is organized as follows. We give an account of related work in Section 2. Section 3 gives all necessary preliminaries, on Horn clauses, languages and recognizability, resolution and its refinements, semilinear sets and branching vector addition systems with states (BVASS).

Once preliminaries have been taken care of, we can define formally what we mean by $\mathcal{E}$-tree automata, whether one-way or two-way, alternating or not, in Section 4. Our interest in such automata stems most particularly from the case where $\mathcal{E}$ is the equational theory AC of finitely many associative commutative symbols $+_i$, $1 \leq i \leq p$. As we have already said, this is justified by the application to group key agreement protocols; we refer to (Goubault-Larrecq et al., 2004) for details. We believe that the theory AC is so pervasive that one/two-way, alternating or not, AC-tree automata will find their way in other applications. A likely application is to XML Schemas, where the theory AC would be used to account for the fact that XML documents are trees whose nodes have a multiset, not a sequence, of sons.

We proceed to show some limitations of AC-tree automata in Section 5. We show mainly that alternation leads to undecidability, already in the constant-only case, where the only function symbols are constants, plus one AC symbol $+$.

Because the constant-only case is, in fact, central to the general case, as will become progressively more apparent in later sections, we show in Section 6 that AC-satisfiability is decidable for non-alternating two-way AC^0-tree automata, i.e., those non-alternating two-way AC-tree automata that arise in the constant-only case. These results rely on the fact that an AC symbol together with constants allows us to encode counters, so that our automata in the constant-only case correspond to various notions of automata working on counters. These include Parikh images of context free grammars, which recognize semilinear sets or Presburger-definable sets (Parikh, 1966; Ginsburg and Spanier, 1966), as well as Petri nets and VASS (Reutenauer, 1993), and their extensions like Branching VASS (Verma and

Goubault-Larrecq, 2005).

We gradually reduce the AC-satisfiability problem for larger and larger classes of two-way AC-tree automata in Section 7, culminating with so-called *AC-standard* two-way AC-tree automata, where so-called $+$-push clauses are restricted to being unconditional (see later for definitions). We show that these classes describe the same languages as ordinary, one-way AC-tree automata, and are therefore closed under union and intersection.

In passing, we show in Section 7.1 that we can always assume without loss of generality that there is exactly one AC function symbol $+$, instead of several.

We prove again that intersection-emptiness is decidable for AC-standard two-way AC-tree automata, using rather different techniques based on resolution techniques, and specifically on the use of *grey oracles*, due to Goubault-Larrecq et al. (2004). This is more technical than previous sections, unfortunately, but has one advantage at least. Since this new technique is not limited to *AC-standard* two-way AC-tree automata, we are able to show that intersection-emptiness for two-way AC-tree automata (not just AC-standard ones) is decidable as soon as it is in the constant-only case. The latter problem is left open. As we shall argue later on, this last remaining open case is likely to be hard, as it includes vast generalizations of problems as difficult as Petri net reachability, to which they do not seem to reduce easily.

We conclude in Section 9.

For quick reference in the AC case, Figure 1 displays a map of the different kinds of AC-tree automata we consider in this paper.

Acknowledgments Thanks to H. Comon and L. Fribourg for many stimulating discussions, to A. Finkel and S. Lasota for helpful comments, to A. Podelski for suggesting that clausal formats may provide a better tool for studying generalized automata. A great many thanks are due to an anonymous STACS'02 referee, who found mistakes in our decidability arguments for intersection-emptiness in the constant-only case, which prompted our branching extension of VASS, BVASS, and the associated notion of Karp-Miller trees (Verma and Goubault-Larrecq, 2005). Finally we thank the two anonymous referees for their technical remarks, pointers to related work, and suggestions for improving the presentation of this, admittedly technical, paper.

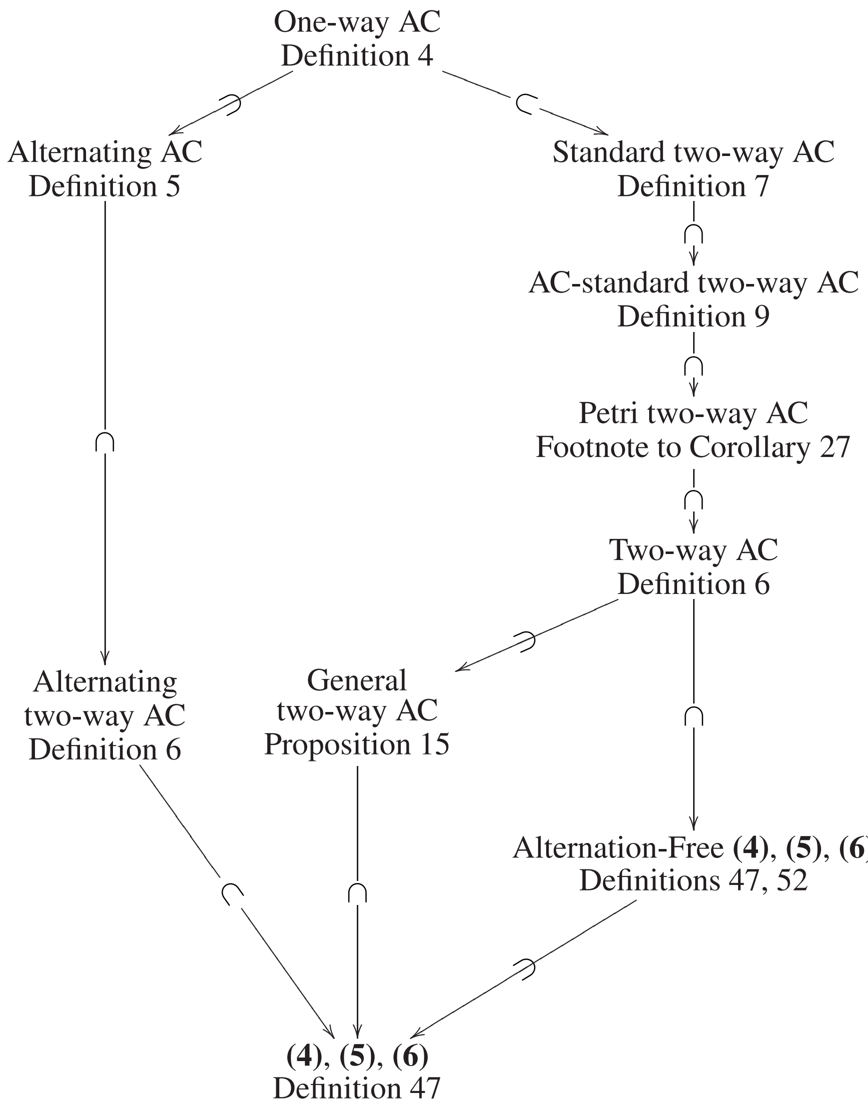


Fig. 1. AC-tree automata considered in this paper

2 Related Work

There is a large literature on finite tree automata, see Comon et al. (1997); Gécseg and Steinby (1997). Applications abound in rewriting and automated theorem proving notably: approximations of reachability sets for rewrite systems (Genet, 1998), disunification and inductive reducibility (Lugiez and Moysset, 1994), unification under constraints (Kaji et al., 1997), ground reducibility (Comon and Jacquemard, 1997), automated inductive theorem proving (Bouhoula and Jouannaud, 1997), fast tree matching (Li, 1988), automated model building in first-order logic (Peltier, 1997), etc. These applications deal with automata on *finite* trees, and this is what we are interested in here. We won't deal with automata on infinite trees (Thomas, 1990), which are also fundamental, e.g. in temporal and program logics (Emerson and Jutla, 1988).

Two-way automata, a.k.a. *pushdown processes*, where transitions may not only construct but also destruct terms, are also classical. The relation with certain Horn sets called *uniform programs* was pioneered in Frühwirth et al. (1991), and refined in e.g., Charatonik and Podelski (1998). *Cartesian approximation* is the key to define upper approximations of various sets of ground atoms, e.g., *success sets*. (While there is no difficulty to do the same in the AC case, getting two-way, alternating AC-tree automata, it is more difficult to get rid of alternation. This is important, as we shall see in Section 5, since alternation causes undecidability in the AC case.)

It is important to distinguish pushdown processes from pushdown *automata* (Schimpf and Gallier, 1985), which recognize the strictly larger class of context-free tree languages. This is why we prefer the phrase *two-way* automata. Conversely, standard automata, where transitions only construct terms, will be called *one-way* automata.

The idea of generalizing tree automata to recognize languages of terms *modulo an equational theory* $\mathcal{E}$ is then natural, and a canonical choice of theory is that of one associative-commutative (AC) symbol $+$. The AC case has been explored a number of times, e.g., by Courcelle (1989); Niehren and Podelski (1993); Lugiez (1998). The general case of so-called *equational automata* has been studied by Ohsaki (2001); Ohsaki and Takai (2002). We shall also deal with this general case, although we emphasize the AC case.

While not all notions of AC-tree automata coincide, there is always a common core. For example, the automata of Lugiez (1998) have additional sort restrictions, but are also extended with a rich constraints language. Recent work by Lugiez (2003) dispenses with the sort restrictions and extends this latter work by considering AC-tree automata with Presburger-definable constraints, catering for an extremely rich framework that includes most proposals of AC-tree automata with decidable emptiness problems until now. Lugiez also shows closure of his class under all Boolean operations. Nonetheless, there is no known reduction of *two-way* AC-tree automata, as studied here, to Lugiez's, and there cannot be any reduction of *alternating* AC-tree automata to Lugiez's, as the former recognize all recursively enumerable sets (Proposition 11) whereas emptiness is decidable for the latter. XML document processing is the main motivation for the automata proposed by Lugiez. Related notions of automata and logics for XML document processing have been proposed by several authors, e.g. Seidl et al. (2003) and Boneva and Talbot (2005). All the papers cited above deal with one-way AC-tree automata.

Ohsaki (2001) investigates a larger framework of so-called *equational tree automata*, modulo some equational theory $\mathcal{E}$. It is difficult to compare these with our $\mathcal{E}$-tree automata. For one, again Ohsaki's automata are not two-way automata; we return to this point below.

Leaving subtleties about two-wayness aside, one might think that *regular equa-*

tional tree automata, a restriction of equational tree automata, also due to Ohsaki, should be the same as our one-way $\mathcal{E}$-tree automata (Section 4.1). Despite the similarities, these are in general different notions. For example, consider the theory $\mathcal{E}$ defined by $f(x, x) = 0$ for every x, and the automaton with two states q_0 and q_1, q_1 being final, and the only transition $f(q_0, q_0) \to q_1$. (In our notation, $q_1(f(X, Y)) \Leftarrow q_0(X), q_0(Y)$, see later.) In particular, no term is recognized at q_0. With our definition, where every term recognized at q_1 must be equal modulo $\mathcal{E}$ to some term of the form $f(u, v)$ where u and v are recognized at q_0, no term is recognized at q_1. With Ohsaki's definition, $f(q_0, q_0)$ is equated with 0 by the theory, so q_1 recognizes the term 0. That is, for Ohsaki, the equational theory $\mathcal{E}$ applies not only to ordinary terms, but also to the fake terms such as $f(q_0, q_0)$ that are used as auxiliaries in defining recognizability. Still, our *one-way* $\mathcal{E}$-tree automata coincide with Ohsaki's *regular* automata when $\mathcal{E}$ is a *linear* theory, in particular in the AC case.

The general form of automata considered by Ohsaki not only has transitions of the form $f(q_1, \ldots, q_n) \to q$, but also of the form $f(q_1, \ldots, q_n) \to f(q'_1, \ldots, q'_n)$, where $q_1, \ldots, q_n, q'_1, \ldots, q'_n, q$ are states. We do not. (Ohsaki's purpose seems to be able to represent $\mathcal{E}$-closures of regular tree languages.) In fact, the second kind of transition $f(q_1, \ldots, q_n) \to f(q'_1, \ldots, q'_n)$ does not have any equivalent in our formulation. Conversely, our push clauses (see Section 4.2) do not seem to be describable in a rewrite rule based notation. The latter entails that we cannot simply reduce our emptiness and intersection-emptiness questions of two-way AC-tree automata to reachability in ground AC-rewrite systems (Mayr and Rusinowitch, 1998), as Ohsaki and Takai (2002) do. To be precise, to show decidability in the AC case, we in fact show how to eliminate the push clauses under various restrictions. This means that these decidable classes of automata correspond to Ohsaki's regular AC-tree automata. However this correspondence is not direct, and holds for specific theories like AC but not for more general theories that we are interested in (Verma, 2003c).

The works by Ohsaki and Lugiez cited above encode AC-tree automata by ground rewrite rules, while we prefer to encode them by sets of Horn clauses. Without any equational theory, the two formulations are well-known to be equivalent. As we discussed above, they diverge in the presence of equational theories. The Horn clause formulation has the advantage that it allows one to write the semantics of the problem at hand, such as modeling cryptographic protocols, directly in logic. Also, alternation and two-wayness are more natural concepts in Horn clause notation.

Our notion of alternation in equational tree automata is close to the conjunction operator in *conjunctive grammars* (Okhotin, 2001). In our terminology, conjunctive grammars are one-way alternating tree automata over a signature consisting of an associative symbol (possibly with a unit) and constants (and no other symbols of larger arity). While we shall mention briefly associative symbols (Proposition 16), our main interest in this paper is the theory of associative-commutative symbols.

However they turn out to have some similar properties. Similarly to the case for alternating AC-tree automata, emptiness for conjunctive grammars is undecidable and membership is decidable.

We shall use techniques related to Petri nets. In particular, we shall use some of Verma and Goubault-Larrecq (2005)'s results on the fact that coverability trees à la Karp-Miller for a *branching* extension of vector addition systems with states (VASS), which were called BVASS there, are finite. BVASS were independently introduced by de Groote et al. (2004), under the name of vector addition tree automata (VATA) to attack the problem whether provability in multiplicative-exponential linear logic was decidable.

3 Preliminaries

Fix a signature Σ of function symbols, each coming with a fixed arity, and let $\mathcal{E}$ be an equational theory, inducing a congruence $\approx_{\mathcal{E}}$ on all terms built on Σ. In this paper, $\mathcal{E}$ will usually be the theory AC of one or several symbols being associative and commutative. We assume that Σ contains at least one constant.

An *atomic formula* (or *atom*) is a pair $P(s)$ of a predicate symbol P, taken from some fixed set $\mathcal{P}$, and of a term s on Σ. (Wlog, we restrict to unary predicate symbols.) A *literal* is either a *positive* literal $+P(t)$, or a *negative* literal $-P(t)$. A *clause* is a disjunction of literals $\pm_1 P_1(t_1) \vee \pm_2 P_2(t_2) \vee \ldots \vee \pm_k P_k(t_k)$. A Horn clause is one containing at most one positive literal: we also write $P(t) \Leftarrow P_1(t_1), \ldots, P_n(t_n)$ for the *definite clause* $+P(t) \vee -P_1(t_1), \ldots, -P_n(t_n)$, and $\perp \Leftarrow P_1(t_1), \ldots, P_n(t_n)$ for the *goal clause* $-P_1(t_1), \ldots, -P_n(t_n)$.

The semantics of clauses is given as usual (Chang and Lee, 1973). A *structure* I is a tuple consisting of a non-empty set D (the *domain*), together with subsets I_P of D, one for each predicate P, and functions $I_f : D^n \to D$ for each function $f \in \Sigma$, of arity n. Given any *environment* ρ mapping variables to elements of D, the *value* $I\,[\![t]\!]\,\rho$ of a term t is defined by $I\,[\![x]\!]\,\rho = \rho(x)$ for every variable x, $I\,[\![f(t_1, \ldots, t_n)]\!]\,\rho = I_f(I\,[\![t_1]\!]\,\rho, \ldots, I\,[\![t_n]\!]\,\rho)$. Then we let $I, \rho \models P(t)$, and say that $P(t)$ *holds* in I, ρ, if and only $I\,[\![t]\!]\,\rho \in I_P$. A clause C holds under I, ρ (and we write $I, \rho \models C$) if and only if $I, \rho \models P(t)$ for some literal $+P(t)$ in C, or $I, \rho \not\models P(t)$ for some literal $-P(t)$ in C.

In the special case of Horn clauses, this can be recast as follows. By convention, let $I, \rho \not\models \perp$. We let $I, \rho \models C$, where C is a Horn clause $A \Leftarrow A_1, \ldots, A_n$, if and only if $I, \rho \not\models A_i$ for some i, $1 \leq i \leq n$, or $I, \rho \models A$.

The structure I is a *model* of the clause C if and only if $I, \rho \models C$ for every environment ρ; we then write $I \models C$. I is a model of a set S of clauses if and only if

$I \models C$ for every clause C in S; we write $I \models S$ for this.

The structure I is an *$\mathcal{E}$-structure* if and only if, whenever s and t are two terms that are equal modulo $\mathcal{E}$, then $I\,[\![s]\!]\,\rho = I\,[\![t]\!]\,\rho$ for every environment $\mathcal{E}$. An $\mathcal{E}$-structure that is a model of C, resp. S, is called an *$\mathcal{E}$-model* of C, resp. S.

We then say that a clause C, resp. a clause set S, is *$\mathcal{E}$-satisfiable* if and only if it has an $\mathcal{E}$-model.

A term, an atom, a literal, a clause is *ground* if and only if it contains no free variable. A *substitution* σ is any map from variables to terms. We write $[x_1 := t_1, \ldots, x_n := t_n]$ the substitution mapping x_i to t_i, $1 \leq i \leq n$, and any other variable to itself. The *domain* $\mathrm{dom}\ \sigma$ of σ is $\{x | x\sigma \neq x\}$. We also write $t\sigma$ the result of applying the substitution σ to the term t: $x\sigma = \sigma(x)$, $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$. An *instance* of t is any term that is equal modulo $\mathcal{E}$ to $t\sigma$, for some substitution σ.

The *Herbrand universe* $H_{\mathcal{E}}$ is the set of all $\mathcal{E}$-equivalence classes of ground terms. A *Herbrand structure* is any structure I whose domain is $H_{\mathcal{E}}$, and such that I_f maps any n-tuple of ground terms t_1, …, t_n modulo $\mathcal{E}$, to the term $f(t_1, \ldots, t_n)$, again modulo $\mathcal{E}$, where f has arity n. It is well-known that a clause set has an $\mathcal{E}$-model if and only if it has an $\mathcal{E}$-Herbrand model, i.e., one which is a Herbrand structure.

Any Herbrand structure I can be alternatively characterized as a set of ground atoms that is stable under $\mathcal{E}$: namely the ground atoms $P(t)$ such that $I \models P(t)$ (the environment part is irrelevant here, hence omitted). In this setting, Herbrand structures can be ordered by inclusion. Then, any $\mathcal{E}$-satisfiable Horn clause set has a *least* Herbrand model. (This is the first place where dealing with *Horn* clauses matters.) In particular, any set of definite clauses has a least Herbrand model; indeed, it has a Herbrand model, which contains every ground atom.

An alternative characterization of least Herbrand models, which should be familiar to Prolog semanticists, is as follows. Fix a Horn clause set S. Let $\mathcal{F}$ be the set of all ground atoms, union $\perp$, and let $\perp\sigma$ be defined as $\perp$, by convention. In other words, we consider $\perp$ as ground, and extend $\approx_{\mathcal{E}}$ so that $t \approx_{\mathcal{E}} \perp$ if and only if $t = \perp$. Define the operator T_S from $\mathbb{P}(\mathcal{F})$ to $\mathbb{P}(\mathcal{F})$ by

$$T_S(I) = \{A' | A \Leftarrow A_1, \ldots, A_n \in S, A' \approx_{\mathcal{E}} A\sigma \text{ ground}, A_1\sigma \in I, \ldots, A_n\sigma \in I\}$$

Since T_S is monotonic with respect to inclusion, it has a least fixpoint. In fact, this least fixpoint is $\bigcup_{n \in \mathbb{N}} T_S^n(\emptyset)$, and is just the least Herbrand model of S in case it does not contain $\perp$. If it contains $\perp$, then S is $\mathcal{E}$-unsatisfiable.

We shall need to decide whether given finite sets of Horn clauses modulo AC or ACU are AC-unsatisfiable or not. Computing $\bigcup_{n\in\mathbb{N}} T_S^n(\emptyset)$ directly is in general not an option, since it will usually be infinite. One well-known technique to decide satisfiability is *resolution* and its refinements, in particular ordered resolution with selection (Bachmair and Ganzinger, 2001).

Let $\succ$ be a strict stable ordering on atomic formulas. By *stable* we mean that if $P(s) \succ Q(t)$, then $P(s\sigma) \succ Q(t\sigma)$ for any substitution σ. Let sel be a function mapping each clause to a subset of its negative literals.

Ordered resolution with selection is the rule that allows one to derive the conclusion (below the bar) provided we have already derived the premises (above):

$$\frac{\begin{array}{cc} C_1 \vee +A_{11} \vee \ldots \vee +A_{1m_1} & \\ \ldots & C' \vee -A'_1 \vee \ldots \vee -A'_n \\ C_n \vee +A_{n1} \vee \ldots \vee +A_{nm_n} & \end{array}}{(C_1 \vee \ldots \vee C_n \vee C')\sigma}$$

where:

(i) $n \geq 1, m_1 \geq 1, \ldots, m_n \geq 1$;
(ii) $\sigma = \mathrm{mgu}(A_{11} \doteq \ldots \doteq A_{1m_1} \doteq A'_1, \ldots, A_{n1} \doteq \ldots \doteq A_{nm_n} \doteq A'_n)$, i.e., σ is the most general unifier (mgu) of the equations $A_{11} \doteq \ldots \doteq A_{1m_1} \doteq A'_1, \ldots, A_{n1} \doteq \ldots \doteq A_{nm_n} \doteq A'_n$;
(iii) for every i, $1 \leq i \leq n$, sel $(C_i \vee +A_{i1} \vee \ldots \vee +A_{in_i}) = \emptyset$ and $A_{i1}, \ldots, A_{in_i}$ are maximal atomic formulae in $C_i \vee +A_{i1} \vee \ldots \vee +A_{in_i}$ with respect to $\succ$;
(iv) sel $(C' \vee -A'_1 \vee \ldots \vee -A'_n) = \{-A'_1, \ldots, -A'_n\} \neq \emptyset$, or sel $(C' \vee -A'_1 \vee \ldots \vee -A'_n) = \emptyset$ and $A'_1, \ldots, A'_n$ are maximal in $C' \vee -A'_1 \vee \ldots \vee -A'_n$ with respect to $\succ$.

For additional definitions, see Bachmair and Ganzinger (2001). It is implicit in the rule above that all premises have been renamed so that no two premises share any free variable. The right premise is called the *main premise*, all others are *side* premises. The conclusion is often called a *resolvent* of the premises.

In the case of Horn clauses, this simplifies to the rule:

$$\frac{A_1 \Leftarrow H_1 \quad \ldots \quad A_n \Leftarrow H_n \quad A \Leftarrow H, A'_1, \ldots, A'_n}{(A \Leftarrow H, H_1, \ldots, H_n)\sigma}$$

where H, H_1, …, H_n are *bodies*, i.e., sets of atomic formulas, comma denotes union of such sets, and the following conditions are met:

(i) $n \geq 1$;
(ii) $\sigma = \text{mgu}(A_1 \doteq A'_1, \dots, A_n \doteq A'_n)$;
(iii) for every i, $1 \leq i \leq n$, sel $(A_i \Leftarrow H_i) = \emptyset$ and A_i is maximal in $A_i \Leftarrow H_i$ with respect to $\succ$;
(iv) letting C be $A \Leftarrow H, A'_1, \dots, A'_n$, sel $(C) = \{-A'_1, \dots, -A'_n\} \neq \emptyset$, or sel $(C) = \emptyset$ and A'_1, …, A'_n are maximal in C with respect to $\succ$.

This rule is *sound*, i.e., every conclusion is a consequence of the premises; in particular, if the empty clause $\perp$ is derivable from a given set of clauses S, then S is unsatisfiable. It is also *complete*: if S is unsatisfiable, then one can derive $\perp$ from S in finitely many steps of ordered resolution with selection.

In passing, choosing sel so that it selects every negative literal (i.e., sel $(A \Leftarrow A_1, \dots, A_n) = \{-A_1, \dots, -A_n\}$) yields the so-called *unit resolution* rule on Horn clauses:

$$\frac{P_1(u_1) \quad \dots \quad P_n(u_n) \quad P(t) \Leftarrow P_1(t_1), \dots, P_n(t_n)}{P(t\sigma)} \; \sigma = \text{mgu}(u_1 \doteq t_1, \dots, u_n \doteq t_n)$$

which is not only sound and complete (in the Horn case), but is also such that $\bigcup_{n \in \mathbb{N}} T^n_S(\emptyset)$ is exactly the set of ground instances of clauses that we can deduce from the set of Horn clauses S by unit resolution. In short, unit resolution computes the least Herbrand model (if any).

While unit resolution, in some sense, derives new facts in a forward manner, *input resolution* derives new goals, working its way backwards:

$$\frac{A_1 \Leftarrow H_1 \quad \dots \quad A_n \Leftarrow H_n \quad \perp \Leftarrow A'_1, \dots, A'_n}{(\perp \Leftarrow H_1, \dots, H_n)\sigma}$$

where $n \geq 1$, $\sigma = \text{mgu}(A_1 \doteq A'_1, \dots, A_n \doteq A'_n)$.

Soundness and completeness hold for each variant of resolution, and in the case of ordered resolution with selection, whatever sel , and whatever the stable ordering $\succ$ is. It is folklore that soundness and completeness still hold when terms are taken modulo some equational theory $\mathcal{E}$, provided σ is taken to be any member of a *complete set of unifiers* (csu) $\text{csu}(A_1 \doteq A'_1, \dots, A_n \doteq A'_n)$ in condition *(ii)*, and $\succ$ is *compatible with* E, meaning that if s_1, s_2 are equal mod E, if t_1, t_2 are equal mod E, and $s_1 \succ t_1$ then $s_2 \succ t_2$. (Implicit here is the fact that we also replace unsatisfiability by $\mathcal{E}$-unsatisfiability.) This was already the case for other refinements of resolution (Plotkin, 1972). Csus always exist, but need not be finite or even computable. One can compute a finite one for the theory of associativity and commutativity (AC), resp. with unit (ACU) (Stickel, 1981; Fages, 1984).

Independently of equational reasoning, soundness and completeness are preserved when tautologies and various forms of subsumed clauses are removed, at any moment (preferably at the earliest) (Bachmair and Ganzinger, 2001). This will be crucial in showing that resolution terminates on various classes of Horn clauses modulo AC, therefore showing decidability of these classes. Equally crucial will be the so-called *splitting* rules. A clause of the form $C \vee C'$, where C and C' are non-empty clauses that share no free variable, is called *splittable*. Given a set of clauses $S \cup \{C \vee C'\}$, where $C \vee C'$ is splittable, the standard version of splitting (Weidenbach, 2001) then considers showing that both $S \cup \{C\}$ and $S \cup \{C'\}$ are unsatisfiable to conclude that $S \cup \{C \vee C'\}$ is. Instead, we shall use Riazanov and Voronkov's special brand of splitting (Riazanov and Voronkov, 2001; Voronkov, 2001), as explained in Goubault-Larrecq et al. (2004), and call it *splittingless splitting* to distinguish it from ordinary splitting. The idea is that when $C \vee C'$ is splittable, then it is equivalent to $\exists q \cdot (q \Rightarrow C) \wedge (\neg q \Rightarrow C')$, where q is a fresh propositional symbol.

$$\frac{C \vee C'}{C \vee -\ulcorner C' \urcorner \qquad +\ulcorner C' \urcorner \vee C'}$$

We make this formal as follows (Roger, 2003; Goubault-Larrecq, 2003). We first define formally what it means to create fresh propositional symbols. Fix a set $\mathcal{P}$ of predicate symbols. A *$\mathcal{P}$-clause* is any clause whose predicate symbols are all from $\mathcal{P}$. These will be our ordinary clauses. Let then $\mathcal{Q}$ be some set of zero-ary predicate symbols disjoint from $\mathcal{P}$, in one-to-one correspondence with the set of $\mathcal{P}$-clauses modulo renaming: for each $\mathcal{P}$-clause C, let $\ulcorner C \urcorner$ be a symbol in $\mathcal{Q}$, so that $\ulcorner C \urcorner = \ulcorner C' \urcorner$ iff there is a renaming ϱ such that $C = C'\varrho$. These will be our fresh symbols q; however notice that we allow ourselves to reuse to same symbol $q = \ulcorner C' \urcorner$ when we meet the same clause C' twice. The rule of *splittingless splitting* is shown on the right, where C and C' are two non-empty subclauses sharing no variable, where C' is restricted to be a $\mathcal{P}$-clause, and C is required to contain at least an atom $P(t)$ with $P \in \mathcal{P}$.

The effect of the rule is to *replace* $C \vee C'$ by the two clauses $C \vee -\ulcorner C' \urcorner$ and $+\ulcorner C' \urcorner \vee C'$ in conclusion. Intuitively, $\ulcorner C' \urcorner$ is a propositional symbol that abbreviates the negation of C', i.e., that is false exactly when C' is valid.

Ordered resolution with selection, using $\mathcal{E}$-unification, is sound and complete, even when splittingless splitting is applied eagerly (i.e., when both rules can be applied, apply splittingless splitting), provided $\succ$ is a stable ordering such that $P(t) \succ q$ for every $P \in \mathcal{P}$, $q \in \mathcal{Q}$. (We say that $\succ$ is *admissible*.) See Goubault-Larrecq et al. (2004) for details.

In the sequel, we shall always use a special form of splittingless splitting, which we call *ϵ-splitting*: this is the special case where C' is a *negative block* $-P_1(x) \vee \ldots \vee -P_n(x)$ ($n \geq 1$; the variable x is the same in each literal), and where $C \vee C'$

is Horn. The ϵ-splitting rule can be reexplained as the one that replaces any clause $A \Leftarrow H, P_1(x), \ldots, P_n(x)$, where x is not free in A or H, by the two clauses $A \Leftarrow H, q$ and $q \Leftarrow P_1(x), \ldots, P_n(x)$, where $q = \ulcorner -P_1(x) \vee \ldots \vee -P_n(x) \urcorner$; in effect, this defines q as being true if and only if there is a term satisfying all of P_1, $\ldots$, P_n in the least Herbrand model (if any exists).

Finally, it is important to note that there is a more synthetic way of writing the unit resolution rule, which is equivalent from the standpoint of derivability of the empty clause $\perp$:

$$\frac{P_1(u_1) \quad \ldots \quad P_n(u_n)}{P(t\sigma)} \; P(t) \Leftarrow P_1(t_1), \ldots, P_n(t_n) \tag{1}$$

where $\sigma \in \mathrm{csu}_{\mathcal{E}}(u_1 \doteq t_1, \ldots, u_n \doteq t_n)$. (Modulo $\mathcal{E}$, recall that we need to replace mgus by csus.) This notation may appeal more to the reader. See e.g., Chang and Lee (1973) where semantic resolution and therefore also hyperresolution and unit resolution are presented in this way.

3.2 Languages, Recognizability

Our impetus in using sets of Horn clauses is to define various forms of automata. For all these notions, the notions of *recognizability*, and of *language* recognized at some state will be the same. Therefore we choose to introduce these notions here.

Given an $\mathcal{E}$-satisfiable set of Horn clauses S, and a predicate symbol P, the *language* $L_P(S)$ of S at *state* P is the set of all $\mathcal{E}$-equivalence classes of ground terms t such that $P(t)$ is in the least Herbrand model of S. The elements of $L_P(S)$ are called the ($\mathcal{E}$-equivalence classes of) terms *recognized* at P in S.

By abuse of language, we say that P is *empty* in S if and only if $L_P(S)$ is empty, and similarly for other properties. We have the following easy lemmas. The first one characterizes recognizability semantically.

Lemma 1 *Given an $\mathcal{E}$-satisfiable set S of Horn clauses, the ground term t is recognized at P in S if and only if S plus the clause $\perp \Leftarrow P(t)$ is $\mathcal{E}$-unsatisfiable.*

PROOF. If $t \in L_P(S)$, then by definition $P(t)$ is in the least Herbrand model of S, so it is in every Herbrand model of S; it follows that S plus $\perp \Leftarrow P(t)$ is $\mathcal{E}$-unsatisfiable. Conversely, if S plus $\perp \Leftarrow P(t)$ is $\mathcal{E}$-unsatisfiable, then every model of S must fail to satisfy $\perp \Leftarrow P(t)$, so must contain $P(t)$; therefore $t \in L_P(S)$. $\square$

The second lemma characterizes emptiness.

Lemma 2 *Given an $\mathcal{E}$-satisfiable set S of Horn clauses, P is empty in S if and only if S plus the so-called* query clause $\perp \Leftarrow P(x)$ *is $\mathcal{E}$-satisfiable.*

PROOF. If P is empty, then the least Herbrand model of S does not contain any ground atom of the form $P(t)$, hence makes $\perp \Leftarrow P(x)$ true.

Conversely, if S plus $\perp \Leftarrow P(x)$ is $\mathcal{E}$-satisfiable, then its least Herbrand model does not contain any ground atom of the form $P(t)$. Since every model of S plus $\perp \Leftarrow P(x)$ is also a model of S, the least Herbrand model of S is included in that of S plus $\perp \Leftarrow P(x)$, hence does not contain any ground atom of the form $P(t)$ either; so P is empty in S. $\square$

The third lemma characterizes *intersection-emptiness*, that is, given finitely many predicate symbols $P_1, \ldots, P_n$, whether $L_{P_1}(S) \cap \ldots \cap L_{P_n}(S)$ is empty. (We say for short that the intersection of $P_1, \ldots, P_n$ is empty in this case.)

Lemma 3 *Given an $\mathcal{E}$-satisfiable set S of Horn clauses, the intersection of P_1, $\ldots$, P_n is empty in S if and only if S plus the so-called* final intersection clause $\perp \Leftarrow P_1(x), \ldots, P_n(x)$ *is $\mathcal{E}$-satisfiable.*

PROOF. The proof is similar. $\square$

3.3 Semilinear Sets, Vector Addition Systems with States, Branching VASS

A *vector addition system with states*, or *VASS* (Reutenauer, 1993), is a counter machine without zero-test. Alternatively, it is a finite automaton where transitions are labeled with two p-tuples of integers $\nu_{\text{in}}, \nu_{\text{out}} \in \mathbb{N}^p$. A *configuration* is a pair comprised of a state P and a p-tuple of natural numbers $\nu \in \mathbb{N}^p$, which we write as an atom $P(\nu)$. If there is a transition from state P_1 to state P, labeled $\nu_{\text{in}}, \nu_{\text{out}}$, then the VASS may evolve from the configuration $P_1(\nu)$ to the configuration $P(\nu - \nu_{\text{in}} + \nu_{\text{out}})$, provided $\nu \geq \nu_{\text{in}}$. It is understood that all operations, in particular $+$ and $\geq$, are computed componentwise.

Formally, we may recast this in the unifying language of Horn clauses as follows. A *VASS* is any finite set of clauses of the form

$$P(\nu) \tag{2}$$
$$P(x + \nu_{\text{out}}) \Leftarrow P_1(x + \nu_{\text{in}}) \tag{3}$$

where $\nu, \nu_{\text{in}}, \nu_{\text{out}} \in \mathbb{N}^p$.

Clauses (2) are called *initial clauses*, and clauses (3) are *transitions*.

Since p will usually be kept fixed, we don't mention it in the definition. This falls into our general format of clauses modulo an equational theory: the signature Σ is comprised of p distinct constants a_1, ..., a_p, plus one constant 0 and one binary function symbol $+$, and the equational theory $\mathcal{E}$ is the theory of the free commutative monoid generated by a_1, ..., a_p, with addition $+$ and unit 0. In other words, $\mathcal{E}$ is the theory ACU stating that $+$ is associative, commutative, and has 0 as unit, on the signature $a_1, \ldots, a_p, +, 0$. Then the term $\sum_{i=1}^{p} n_i a_i$ represents the vector $(n_1, \ldots, n_p)$.

A VASS $\mathcal{V}$ where every atom $P(t)$ uses the same predicate P is called a *Petri net*. The *places* are the integers i, $1 \leq i \leq p$, or equivalently the p distinct constants a_1, ..., a_p. The *markings* are the p-tuples $\nu \in \mathbb{N}^p$. If $P(\nu)$ is a clause (2) in $\mathcal{V}$, then ν is called an *initial marking* of $\mathcal{V}$. The *transitions* are the clauses of the form (3), which accords with our definition above.

Since we are only interested in the language recognized by a VASS, in the sense of Section 3.2, that is in the sets of ground unit clauses $P(\nu)$ deducible from a VASS by unit resolution, we may without loss of generality assume that, in transitions (3), for every i, $1 \leq i \leq p$, the ith component of ν_{in} and the ith component of ν_{out} are not both non-zero. Then, letting δ be the vector $\nu_{\text{out}} - \nu_{\text{in}}$ in $\mathbb{Z}^p$, there is no ambiguity in writing such clauses

$$P(x + \delta) \Leftarrow P_1(x) \tag{4}$$

understanding that unit resolution with the ground unit clause $P_1(\nu)$ generates $P(\nu + \delta)$, provided $\nu + \delta \in \mathbb{N}^p$. This is in particular, up to the representation of transitions as clauses, the definition used by Reutenauer (1993).

Given any finite sets A and $B = \{\nu_1, \ldots, \nu_k\}$ of vectors in $\mathbb{N}^p$, the smallest set $L_{A,B}$ containing A and such that $\nu \in L_{A,B}$ and $\nu' \in B$ imply $\nu + \nu' \in L_{A,B}$, can also be described as the set of all vectors $\nu_0 + \sum_{i=1}^{k} n_i \nu_i$, $\nu_0 \in A$, $n_1, \ldots, n_k \in \mathbb{N}$. A *linear set* is any set of the form $L_{A,B}$, and a *semilinear set* is any finite union of linear sets. If every δ in clauses (4) is in $\mathbb{N}^p$, i.e., consists of non-negative integers, in other words if transitions (3) are such that $\nu_{\text{in}} = 0$, then it is clear that the languages of each predicate P in any VASS are semilinear sets. This is an instance of Parikh (1966)'s Theorem, see below. Conversely, every semilinear set can be described as the language of P in some collection of clauses (2) and (3) with $\nu_{\text{in}} = 0$. In particular every semilinear set is recognized by some (computable) VASS. The converse fails, as shown by Hopcroft and Pansiot (1979), when $p \geq 5$.

The semilinear sets are closed under intersection, union, complementation, and projection. This is the fundamental observation behind Ginsburg and Spanier (1966)'s Theorem that the semilinear sets are exactly the Presburger-definable sets, i.e., the sets of p-tuples of natural numbers definable as those satisfying some formula of Presburger arithmetic with p free variables.

Another fundamental result is Parikh (1966)'s Theorem. Recall that the commutative image of a string built from symbols $a_1, \ldots, a_p$ refers to the vector $(n_1, \ldots, n_p)$ where n_i is the number of occurrences of a_i in the string. The commutative image of a set of strings is the set of commutative images of its members. Parikh's theorem states that the commutative image of any context-free language is a semilinear set. This result is effective in the sense that, given a context-free grammar G, we can compute a finite family of finite sets A_i, B_i such that the commutative image of the language produced by G is $\bigcup_i L_{A_i,B_i}$. Parikh's Theorem also states that every semilinear set can be realized as the commutative image of some regular set.

One extension of VASS that we shall require here is *branching VASS*, or *BVASS*. They were introduced in Verma (2003a); Verma and Goubault-Larrecq (2005), precisely to solve the problems we present here. Since then de Groote et al. (2004) invented independently the same concept, under the name of *vector addition tree automata* (VATA), and showed that provability in the multiplicative-exponential fragment of linear logic (MELL) was equivalent to reachability in VATA/BVASS. A *BVASS* is a finite set of initial clauses (2), of transitions (3), and of *addition clauses* of the form

$$P(x + y) \Leftarrow P_1(x), P_2(y) \tag{5}$$

where P, P_1, P_2 are predicate symbols.

If in all transitions (3) we have $\nu_{\mathrm{in}} = 0$, then BVASS are nothing else but Parikh images of context-free languages (Verma and Goubault-Larrecq, 2005), and therefore define just the semilinear sets, by Parikh's Theorem. Otherwise, BVASS generalize Petri nets and VASS. It is unknown whether this generalization is proper.

The *covering problem* for VASS or BVASS is, given a VASS or BVASS S and a ground atom $P(\nu)$, whether there is a ground atom $P(\nu_1)$ deducible from S such that $\nu_1 \geq \nu$. (We say $P(\nu)$ can be *covered* in S.) The VASS or BVASS S is *bounded* if and only if there are only finite ground atoms deducible from S. A place i is *bounded* in S if the set of ith components $\nu[i]$ of vectors ν such that $P(\nu)$ is deducible from S is finite. These properties can be decided for VASS easily enough by noting that VASS are well-structured transition systems (Finkel and Schnoebelen, 2001). While BVASS are not even transition systems at all, a similar technique that computes coverability sets backwards allows one to decide coverability similarly: see Goubault-Larrecq and Verma (2002, Lemma 5).

A more complex technique, in the VASS case, is the use of the Karp-Miller coverability tree (Karp and Miller, 1969), which computes a set $KM(S)$ of *generalized atoms* $P(\nu')$—generalized in the sense that $\nu' \in (\mathbb{N} \cup \{+\infty\})^p$—such that any ground atom $P(\nu)$ can be covered in the VASS S if and only if $KM(S)$ contains some $P(\nu')$ with $\nu' \geq \nu$. Moreover, $KM(S)$ is finite and computable. This is because the elements of $KM(S)$ are the labels of the Karp-Miller coverability tree, which is itself finite and computable.

The main result of Verma and Goubault-Larrecq (2005) is to extend this construction to BVASS. To be precise, we have proved in op.cit. that, for any BVASS S, there is a finite set $KM(S)$ (obtained from a generalization of Karp-Miller coverability tree to the case of BVASS) such that:

(1) For every ground atom $P(\nu)$ derivable from S, there is a generalized atom $P(\nu')$ in $KM(S)$ such that $\nu'[i] = \nu[i]$ whenever $\nu'[i] \neq \infty$.
(2) For every generalized atom $P(\nu')$ in $KM(S)$, there is a ground atom $P(\nu)$ deducible from S such that $\nu[i] = \nu'[i]$ whenever $\nu'[i] \neq \infty$. Moreover, we may choose $\nu[i]$ as large as we wish—exceeding any prescribed bound $K \in \mathbb{N}$—for every i such that $\nu'[i] = \infty$.
(3) Finally, $KM(S)$ is finite and computable.

The first two items allow one to decide whether a given VASS S is bounded (check that no ∞ sign occurs in any generalized atom of $KM(S)$), and to decide the covering problem: for any fixed ground atom $P(\nu)$, there exists a ground atom $P(\nu_1)$ deducible from S with $\nu_1 \geq \nu$ if and only if there is a generalized atom $P(\nu')$ in $KM(S)$ such that $\nu' \geq \nu$. In the sequel, we shall in fact need more than just the fact that boundedness and coverability are decidable, and we require to be able to compute the set $KM(S)$ itself.

The *reachability problem* is, given a VASS or BVASS S and a ground atom $P(\nu)$, whether $P(\nu)$ is deducible from S. This problem is decidable for VASS, by the Mayr-Kosaraju algorithm Mayr (1984); Kosaraju (1982); Sacerdote and Tenney (1977); Lambert (1992); see Reutenauer (1993) for a nice and detailed exposition. This algorithm is non-trivial, and of unknown complexity. The best known lower bound is that the problem is EXPSPACE-hard (Mayr, 1984; Lipton, 1976). One of the ingredients in the decision algorithm is the Karp-Miller coverability tree. Even though the latter generalizes to BVASS, it is still unknown whether reachability is decidable for BVASS (and in particular whether MELL provability is decidable).

4 Alternating Two-Way $\mathcal{E}$-Tree Automata

While tree automata recognize sets of terms on some signature Σ, $\mathcal{E}$-tree automata are meant to recognize sets of equivalence classes of terms modulo $\mathcal{E}$. In particular, when $\mathcal{E}$ is the empty theory, we shall retrieve the standard notion of tree automata, whether one-way (the usual kind), alternating, or two-way. We start with one-way, i.e., run-of-the-mill $\mathcal{E}$-tree automata (Definition 4), and work our way towards the more complicated notions like alternating automata (Definition 5) and two-way automata (Definition 6), with or without alternation. To obtain decidability in the case of two-way AC automata, the push clauses involving AC symbols need to be further restricted, which leads us to define AC-standard two-way AC-tree automata (Definition 9), which is the most general form of automata for which we show decidability in this paper.

4.1 One-Way $\mathcal{E}$-Tree Automata

Definition 4 (One-Way $\mathcal{E}$-Tree Automata) *An* one-way $\mathcal{E}$-tree automaton*, or* $\mathcal{E}$-tree automaton *for short, S, is a finite set of clauses of the form:*

$$P(f(x_1, \ldots, x_n)) \Leftarrow P_1(x_1), \ldots, P_n(x_n) \tag{6}$$

$$P(x) \Leftarrow P'(x) \tag{7}$$

where $f \in \Sigma$ and P, P_1, …, P_n, P' are elements of a finite set of unary predicate symbols called the states *of the automaton, and x_1, …, x_n are distinct variables in (6).*

Clauses (6) are called pop clauses*, and clauses (7) are* ϵ-clauses.

This definition does not depend on $\mathcal{E}$. However, we shall always understand the semantics of $\mathcal{E}$-tree automata as that given in Section 3. In other words, we say "$\mathcal{E}$-tree automaton" to stress the fact that they will always be understood modulo $\mathcal{E}$.

The pop clauses (6) are ordinary tree automata transitions. Intuitively, (6) reads as "if x_1 is recognized at state P_1, and …, and x_n is recognized at state P_n, then $f(x_1, \ldots, x_n)$ is recognized at state P". The ϵ-clauses (7) similarly correspond to epsilon transitions. A more thorough discussion of tree automata as clauses can be found in Goubault-Larrecq (2002) or in Frühwirth et al. (1991).

The restriction that x_1, …, x_n should be distinct variables in pop clauses (6) is to avoid technical problems in the sequel. Allowing repeated variables poses no problem in the case of tree automata (i.e., when $\mathcal{E}$ is the empty theory): using repeated variables, as in $P(f(x, x)) \Leftarrow P_1(x)$, would allow us to deal with tree automata *with equality constraints between brothers* (Bogaert and Tison, 1992).

The careful reader will have noticed that we have not defined any initial or final states here. As far as initial states are concerned, they are useless in tree automata, since 0-ary transitions cater for them; i.e., pop clauses of the form $P(c) \Leftarrow$, where c is a 0-ary function symbol (a *constant*) just defines P as an initial state. We have chosen to let final states be specified independently of automata, because this is more versatile in proofs. On the other hand, this shall force us to talk of "state P

being empty in automaton S", instead of just saying that S is empty. If some state is explicitly specified as being final then the language recognized by the automaton will be the set of terms recognized at the final state. Having only one final state instead of many causes no loss of expressiveness for the automata classes that we are interested in.

Given a predicate symbol (a state) P, the language of all terms recognized at P in an $\mathcal{E}$-tree automaton S is already defined: see Section 3.2, and specialize the notions defined there to $\mathcal{E}$-tree automata.

Some readers may have read other definitions of languages of terms recognized at states P. One of the most common goes as follows. A *run* of a term t against the tree automaton S is a tree, whose nodes are labeled by pairs (P, t)—let us write them $P(t)$ for convenience—, where P is a state and t is a ground term, and such that every node $P(f(t_1, \ldots, t_n))$ in the run has n sons $P_1(t_1)$, ..., $P_n(t_n)$, where $P(f(x_1, \ldots, x_n)) \Leftarrow P_1(x_1), \ldots, P_n(x_n)$ is a transition (a pop clause) of the tree automaton S. Then t is *recognized* at P in S provided there exists a run with root (P, t).

A run is then just a derivation using rules of the form:

$$\frac{\begin{matrix}\vdots & & \vdots \\ P_1(t_1) & \ldots & P_n(t_n)\end{matrix}}{P(f(t_1, \ldots, t_n))} \; P(f(x_1, \ldots, x_n)) \Leftarrow P_1(x_1), \ldots, P_n(x_n)$$

But this is just the unit resolution format; see Section 3.1. Conversely, any ground atom $P(t)$ derivable by unit resolution (in particular, under the form (1)) from S is clearly the root of a run. One may rightly claim that runs *are* unit resolution derivations from the clauses defining the automaton S.

4.2 Alternating, Two-Way $\mathcal{E}$-Tree Automata

Frühwirth et al. (1991) note in particular that so-called reduced regular unary-predicate programs, which generalize pop clauses and ϵ-clauses properly in case $\mathcal{E}$ is the empty theory, can be viewed as alternating tree automata (Slutzki, 1985).

Following this insight, let us define:

Definition 5 (Alternating $\mathcal{E}$-Tree Automata) *An* alternating $\mathcal{E}$-tree automaton *is any finite collection of pop clauses (6), of ϵ-clauses (7), and of* intersection clauses *of the form:*

$$P(x) \Leftarrow P_1(x), \ldots, P_n(x) \tag{8}$$

where $n \geq 2$.

Note that intersection clauses are more powerful than final intersection clauses. The latter allow us merely to check intersection-emptiness of ordinary, i.e. non-alternating, automata. While intersection clauses are natural indeed, we shall see that they cause some trouble in alternating AC-tree automata, making the emptiness problem undecidable (Proposition 11). This is also why we shall be interested in intersection-emptiness (see Lemma 3): in the presence of intersection clauses, intersection-emptiness would reduce to emptiness, but not so without them.

Another generalization of tree automata is two-wayness. We use here a definition that suits our needs, but is not entirely like usual definitions of two-way automata (Shepherdson, 1959). Two-wayness can be defined elegantly using clauses, as was pioneered in Frühwirth et al. (1991). This form of two-wayness is crucial in applications to cryptographic protocols (Goubault-Larrecq et al., 2004). To take a typical example, here are the clauses describing what a Dolev-Yao intruder may know relative to the use of (symmetric) encryption `crypt`:

$$I(\mathtt{crypt}(M, K)) \Leftarrow I(M), I(K)$$
$$I(M) \Leftarrow I(\mathtt{crypt}(M, K)), I(K)$$

The first clause states that if the intruder knows M and the key K, he knows (can deduce) the ciphertext $\mathtt{crypt}(M, K)$ ("M encrypted with K"); this is a pop clause. The second clauses states the converse, that the intruder may decrypt: if the intruder knows some ciphertext $\mathtt{crypt}(M, K)$ and the appropriate key K, then he knows the plaintext M. This is a push clause, as defined below.

Definition 6 (Two-Way, Alternating Two-Way $\mathcal{E}$-Tree Automata) *A* two-way $\mathcal{E}$-tree automaton *is any finite set of pop clauses (6), of ϵ-clauses (7), and of* push clauses *of the form:*

$$P_i(x_i) \Leftarrow P(f(x_1, \ldots, x_n)), P_{i_1}(x_{i_1}), \ldots, P_{i_k}(x_{i_k}) \tag{9}$$

where $1 \leq i \leq n$, $1 \leq i_1, \ldots, i_k \leq n$, *and* $i \notin \{i_1, \ldots, i_k\}$.

Similarly, an alternating, two-way $\mathcal{E}$-tree automaton *is any finite set of pop clauses (6), of ϵ-clauses (7), of intersection clauses (8), and of push clauses (9).*

Just like pop clauses (6) can be used to construct new terms $f(x_1, \ldots, x_n)$ recognized at P from terms x_1 recognized at P_1, …, x_n recognized at P_n, push clauses (9) *destruct* terms. An intuitive reading of (9) is: "if $f(x_1, \ldots, x_n)$ is recognized at P, and x_{i_1} is recognized at P_{i_1}, and … and x_{i_k} is recognized at P_{i_k}, then x_i is recognized at P_i".

If $k = 0$ in (9), then we call this a *standard push clause*; otherwise, call this a *conditional* push clause. More precisely:

Definition 7 (Standard, Conditional Push Clauses) *A* standard push clause *is any clause of the form:*

$$P_i(x_i) \Leftarrow P(f(x_1, \ldots, x_n)) \tag{10}$$

where $1 \leq i \leq n$*. A* conditional push clause *is any clause (9) with* $k \neq 0$*, i.e., a push clause that is not a standard push clause.*

Accordingly, a standard (resp. alternating) two-way $\mathcal{E}$-tree automaton $\mathcal{A}$ *is such that every push clause of* $\mathcal{A}$ *is standard.*

Given any set $\mathcal{F}$ *of function symbols, we say that* $\mathcal{A}$ *is* $\mathcal{F}$-standard *if and only if for each* $f \in \mathcal{F}$*, push clauses of the form (9) are standard.*

A standard push clause (10) would be written, using the notations of set constraints, as $f_{(i)}^{-1}(P) \subseteq P_i$, stating that the set of terms t_i such that $f(t_1, \ldots, t_n)$ is recognized at P, for some $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$, is contained in the set of terms recognized at P_i.

We end this tour of $\mathcal{E}$-tree automata by discussing the side-conditions on push clauses, namely $1 \leq i \leq n$, $1 \leq i_1, \ldots, i_k \leq n$. and $i \notin \{i_1, \ldots, i_k\}$. This means that the variable x_i on the left-hand side can only be used in the atom $P(f(x_1, \ldots, x_n))$ but nowhere else on the right-hand side. Another presentation of push clauses is

$$P_i(x_i) \Leftarrow P(f(x_1, \ldots, x_n)), B_1(x_1), \ldots, B_{i-1}(x_i), B_{i+1}(x_{i+1}), B_n(x_n)$$

where $B_j(x_j)$ denotes any finite conjunction $P_{j1}(x_j), \ldots, P_{jn_j}(x_j)$, for each j. Note that we explicitly exclude having some conjunction $B_i(x_i)$ on the right hand side. Not doing this, that is, allowing for the following more general kind of push clause, which we call *general push clauses*,

$$P_i(x_i) \Leftarrow P(f(x_1, \ldots, x_n)), B_1(x_1), \ldots, B_n(x_n) \tag{11}$$

is equivalent, or so we claim, at least in case of the theory AC of one or more associative and commutative symbols, to allowing for push clauses (9) plus intersection clauses, provided there is at least one function symbol of arity one in Σ.

Indeed, it is clear that (11) can be encoded as

$$\begin{aligned} P_i(x_i) &\Leftarrow q(x_i), B_i(x_i) \\ q(x_i) &\Leftarrow P(f(x_1, \ldots, x_n)), B_1(x_1), \ldots, B_{i-1}(x_{i-1}), B_{i+1}(x_{i+1}), \ldots, B_n(x_n) \end{aligned}$$

by introducing a fresh predicate symbol q. The first clause is an intersection clause, and the second clause is a push clause of the form (9). Conversely, any intersection clause $P(x) \Leftarrow P_1(x), P_2(x)$ can be encoded using general push clauses as

$$\begin{aligned} q(f(x)) &\Leftarrow P_2(x) \\ P(x) &\Leftarrow q(f(x)), P_1(x) \end{aligned}$$

where q is a fresh predicate symbol, and f is some function symbol of arity 1; we let the reader show that the case of intersection clauses (8) with $n > 2$ reduces to the case $n = 2$. This encoding works provided the theory $\mathcal{E}$ is such that for any terms s and t, if $f(s) = f(t)$ then $s = t$. This is clearly so for the theory AC of one or more associative and commutative symbols.

In other words, using the general format (11) for push clauses would reintroduce the intersection clauses (8) in disguise.

4.3 AC-Tree Automata

We shall deal specifically in this paper with the following equational theory AC.

Definition 8 (AC) *The theory* AC *is defined on signatures* Σ *that can be split in so-called* AC symbols $+_1$, ..., $+_p$, *the remaining symbols being called* free function symbols*; AC is the theory of associativity and commutativity of* $+_1$, ..., $+_p$, *i.e., the theory axiomatized by:*

$$s +_i (t +_i u) = (s +_i t) +_i u \qquad s +_i t = t +_i s$$

for every i, $1 \leq i \leq p$.

Accordingly, we have the notions of AC-tree automata, two-way AC-tree automata, standard two-way AC-tree automata, etc. Recall that we have also defined $\mathcal{F}$-standard two-way AC-tree automata (Definition 7). Letting the symbols in $\{+_1, \ldots, +_p\}$ be AC and those in $\Sigma \setminus \{+_1, \ldots, +_p\}$ be free, and specializing Definition 7, we get:

Definition 9 (Free-Standard, AC-Standard Two-Way AC-Tree Automata) *A* $+_i$-push clause *(*$1 \leq i \leq p$*) is a push clause (9) with* $f = +_i$*, i.e., of one of the forms*

$$\begin{aligned} &P_1(x_1) \Leftarrow P(x_1 +_i x_2) \\ &P_2(x_2) \Leftarrow P(x_1 +_i x_2) \\ &P_1(x_1) \Leftarrow P(x_1 +_i x_2), P_2^1(x_2), \ldots, P_2^k(x_2) \end{aligned}$$

$$P_2(x_2) \Leftarrow P(x_1 +_i x_2), P_1^1(x_1), \ldots, P_1^k(x_1)$$

where the first two are standard, and the last two are conditional.

A two-way (resp. two-way, alternating) AC-tree automaton is AC-standard *if and only if all $+_i$-push clauses, $1 \leq i \leq p$, are standard.*

A free-push clause *is a push clause (9):*

$$P_i(x_i) \Leftarrow P(f(x_1, \ldots, x_n)), P_{i_1}(x_{i_1}), \ldots, P_{i_k}(x_{i_k})$$

where f is free.

A two-way (resp. two-way, alternating) AC-tree automaton is free-standard *if and only if all free-push clauses, $1 \leq i \leq p$, are standard.*

AC-standard two-way AC-tree automata will be the largest class of automata on which we shall obtain decidability results in this paper.

We briefly describe how the ACU case can be reduced to the AC case, where ACU is the theory where some or all symbols $+_i$ additionally have a unit $\mathbf{0}_i$. First create fresh states $zero_i$ and add clauses $zero_i(\mathbf{0}_i)$ and $zero_i(x +_i y) \Leftarrow zero_i(x), zero_i(y)$ for all symbols $+_i$. For every other state q, add clauses $q(x +_i y) \Leftarrow q(x), zero_i(y)$ for every $+_i$. For every clause of the form $P(x +_i y) \Leftarrow P_1(x), P_2(y)$, add clauses $P(x) \Leftarrow P_1(x), P_2(y), zero_i(y)$ and $P(x) \Leftarrow P_2(x), P_1(y), zero_i(y)$. The intuition is that for every state q in the ACU automaton an atom $q(t)$ is derivable iff $q(t')$ is derivable for every t' obtained from t by successive replacements of subterms $s +_0 \mathbf{0}_i$ by s and of subterms s by $s +_i \mathbf{0}_i$. The clause $P(x) \Leftarrow P_1(x), P_2(y), zero_i(y)$ can be thought of as ϵ-clause $P(x) \Leftarrow P_1(x)$ together with intersection emptiness test on states P_2 and $zero_i$. As we will show intersection-emptiness to be decidable for AC-standard two-way AC-tree automata, hence such clauses do not increase expressiveness and can be effectively eliminated.

Other interesting equational theories are those of Abelian groups (AG), which extends ACU by requiring that every element have an inverse; and the theory ACUX of ACU symbols $+_i$ such that $t +_i t = \mathbf{0}_i$, which extends AG. The latter is in fact the theory of the bitwise *exclusive or* operation, which has independent interest, already in cryptographic protocol verification. See Verma (2003c,b, 2004) for results on the latter theories; let us just say that the AG and ACUX theories are simpler to deal with than the AC and ACU cases.

We shall also sometimes mention the theory A of associativity alone, and the theory ACUI extending ACU with the idempotence axiom $t +_i t = t$. While AC is the theory of non-empty finite multisets, and ACU is the theory of finite multisets, ACUI is the theory of finite sets, with $+_i$ as union. Note that ACUX is also the theory of finite sets, however with $+_i$ as symmetric difference.

5 Undecidability Results

The purpose of this section is to enumerate a few cases where emptiness is undecidable for (resp. alternating, two-way) $\mathcal{E}$-tree automata. The stress is put on the theory AC, but we also consider ACU and AG, the theory of Abelian groups. The main lesson to be learnt here is that alternation causes undecidability.

Let $\mathcal{E}$ be an equational theory on some signature Σ containing a symbol $+$, such that $\mathcal{E}$ entails that $+$ is associative and commutative. For any $n \in \mathbb{N}$, $n \geq 1$, and any term t, write nt for $t + \ldots + t$, where t occurs n times. Write $\sum_{i=1}^{k} n_i t_i$ for the sum $n_1 t_1 + \ldots + n_k t_k$, where it is assumed that $k \geq 1$ and $n_i \geq 1$ for each i, $1 \leq i \leq k$.

Definition 10 (Torsion-Free) *An equational theory $\mathcal{E}$ where $+$ is associative and commutative is* torsion-free *w.r.t. pairwise distinct constants $a_1, \ldots, a_k$ iff $\sum_{i=1}^{k} n_i a_i = \sum_{i=1}^{k} n'_i a_i$ implies $n_i = n'_i$ for every i, $1 \leq i \leq k$.*

The point is that torsion-free theories allow one to encode tuples $(n_1, \ldots, n_k)$ as sums $\sum_{i=1}^{k} n_i a_i$ in a one-to-one manner. The theories AC, ACU, AG are torsion-free; ACUX and ACUI are not. The above definition gives us the flexibility to choose the constants a_i. For example the constant 0 which is unit of $+$ should not be considered here.

Proposition 11 *Let $\mathcal{E}$ be any theory, with an associative-commutative symbol $+$, which is torsion-free w.r.t. four constants. Emptiness is undecidable for alternating (one-way) $\mathcal{E}$-tree automata.*

PROOF. We use a reduction from the emptiness problem for r.e. sets. For every r.e. set E, there is a two-counter machine M (with counters R_1, R_2) such that M accepts, starting with $R_1 = 0$, exactly when the initial value of R_2 is in E. It then suffices to encode configurations of M that lead to acceptance using alternating $\mathcal{E}$-tree automata.

Recall that a two-counter machine (Minsky, 1961) is a finite labeled transition system with an initial state q_0, a final (acceptance) state q_f, and transitions $q \xrightarrow{a} q'$ where a may be Inc R_i, Dec R_i or Zero R_i, $i \in \{1, 2\}$. Inc R_i increments R_i, Dec R_i checks whether R_i is ≥ 1, and if so decrements R_i, and Zero R_i checks whether $R_i = 0$.

A *configuration* of the machine M is a triple (q, m, n) where q is a state, $m, n \in \mathbb{N}$ are the values of R_1 and R_2 respectively.

We then use an encoding similar to that of Ibarra et al. (2001), except that the direction of computation is reversed. By a remark of op.cit., three constants actually suffice for this Proposition. We shall describe it using four, and let the reader do the exercise of realizing why one of them is not necessary. Let a^i_j, $1 \le i, j \le 2$, be the four constants in the statement of the proposition. Configurations (q, m, n) of the two-counter machine are encoded as ground atoms $q((m+x)a^1_1 + xa^2_1 + (n+y)a^1_2 + ya^2_2)$ where $x, y \ge 1$. Incrementing R_1 will be simulated by adding a^1_1, while decrementing it will be simulated by adding a^2_1, and similarly for R_2. The encoding is not one-to-one: e.g., the values x, y in the above encoding may be any positive numbers. However we will ensure that at least one such atom is deducible corresponding to each configuration of the two-counter machine.

Introduce the clauses in Figure 2, where $is\_a^1_1$, $is\_a^1_2$, $is\_a^2_1$, $is\_a^2_2$, $r_{0,0}$, ..., are predicate symbols distinct from all states, and $j \in \{1, 2\}$ Also, with each state q

Predicate	defined by	recognizes:
$is\_a^j_i$	$is\_a^j_i(a^j_i)$	just a^j_i
$zero_i$	$zero_i(x+y) \Leftarrow is\_a^1_i(x), is\_a^2_i(y)$	$na^1_i + na^2_i$, $n \ge 1$
	$zero_i(x+y) \Leftarrow one_i(x), is\_a^2_i(y)$	
one_i	$one_i(x+y) \Leftarrow is\_a^1_i(x), zero_i(y)$	$(n+1)a^1_i + na^2_i$, $n \ge 1$
$r_{0,0}$	$r_{0,0}(x+y) \Leftarrow zero_1(x), zero_2(y)$	$ma^1_1 + ma^2_1 + na^1_2 + na^2_2$, $m, n \ge 0$
nn_i	$nn_i(x) \Leftarrow zero_i(x)$	$(n+p)a^1_i + pa^2_i$,$n \ge 0, p \ge 1$
	$nn_i(x+y) \Leftarrow is\_a^1_i(x), nn_i(y)$	
$state$	$state(x+y) \Leftarrow nn_1(x), nn_2(y)$	$(m+p)a^1_1 + pa^2_1 + (n+q)a^1_2 + qa^2_2$, $m, n \ge 0$,$p, q \ge 1$
st^+_1	$st^+_1(x+y) \Leftarrow is\_a^1_1(x), state(y)$	$(m+p)a^1_1 + pa^2_1 + (n+q)a^1_2 + qa^2_2$,$n \ge 0$,$m, p, q \ge 1$
st^+_2	$st^+_2(x+y) \Leftarrow is\_a^1_2(x), state(y)$	$(m+p)a^1_1 + pa^2_1 + (n+q)a^1_2 + qa^2_2$, $m \ge 0$, $n, p, q \ge 1$
st^0_1	$st^0_1(x+y) \Leftarrow zero_1(x), nn_2(y)$	$pa^1_1 + pa^2_1 + (n+q)a^1_2 + qa^2_2$, $n, p, q \ge 0$
st^0_2	$st^0_2(x+y) \Leftarrow nn_1(x), zero_2(y)$	$(m+p)a^1_1 + pa^2_1 + qa^1_2 + qa^2_2$, $m, p, q \ge 0$

Fig. 2. Auxiliary clauses used in encoding two-counter machines

of M, associate two fresh predicate symbols q^+_1 and q^+_2, distinct from each other, from every state, and from every predicate introduced above. Add the intersection clauses $q_i^+(x) \Leftarrow q(x), st_i^+(x)$ for $i \in \{1,2\}$; q_i^+ recognizes every configuration recognized by q such that R_i is not zero. We translate the machine M as follows:

(1) Acceptance: $q_f(x) \Leftarrow state(x)$.

(2) $q \xrightarrow{a} q'$, $a = $ Inc R_i: $\qquad q(x+y) \Leftarrow is\_a_i^2(x), {q'}_i^+(y)$.

(3) $q \xrightarrow{a} q'$, $a = $ Dec R_i: $\qquad q(x+y) \Leftarrow is\_a_i^1(x), q'(y)$.

(4) $q \xrightarrow{a} q'$, $a = $ Zero R_i: $\qquad q(x) \Leftarrow q'(x), st_i^0(x)$.

Let S be the set of clauses thus obtained. We have the following two claims:

Claim 12 *If (q,m,n) is a configuration of M that leads to acceptance, i.e., to some configuration (q_f, m', n'), then for some $N \geq 1$, the atom $q((m+x)a_1^1 + xa_1^2 + (n+y)a_2^1 + ya_2^2)$ is deducible from S by positive unit resolution for all $x, y \geq N$.*

PROOF. We do induction on the number of moves made by the machine from the configuration (q,m,n) to lead to acceptance. If the number of moves is zero then it means $q = q_f$ hence we can use the clause $q_f(x) \Leftarrow state(x)$ to deduce all atoms of the form $q_f((m+x)a_1^1 + xa_1^2 + (n+y)a_2^1 + ya_2^2)$ for $x, y \geq 1$. Hence $N = 1$ satisfies the requirements. The main interesting case is when the machine makes an increment move from the configuration (q,m,n). Suppose it increments R_1 to go to configuration $(q', m+1, n)$ which leads to acceptance. By induction hypothesis we have some $N' \geq 1$ such that $q'((m+1+x)a_1^1 + xa_1^2 + (n+y)a_2^1 + ya_2^2)$ is deducible for all $x, y \geq N'$. Then ${q'}_1^+((m+1+x)a_1^1 + xa_1^2 + (n+y)a_2^1 + ya_2^2)$ is also deducible for all $x, y \geq N'$. We use the clause $q(x+y) \Leftarrow is\_a_1^2(x), {q'}_1^+(y)$ to deduce $q((m+1+x)a_1^1 + (x+1)a_1^2 + (n+y)a_2^1 + ya_2^2)$ for all $x, y \geq N'$. Hence by letting $N = N'+1$ we see that we can deduce atoms $q((m+x)a_1^1 + xa_1^2 + (n+y)a_2^1 + ya_2^2)$ for all $x, y \geq N$. $\square$

Claim 13 *All unit clauses deducible from S by positive unit resolution of the form $q(t)$ are such that t is a ground term of the form $(m+x)a_1^1 + xa_1^2 + (n+y)a_2^1 + ya_2^2$, for some $x, y \geq 1$, where (q,m,n) leads to acceptance in M.*

The first claim means that, although we may not deduce all representatives of the configurations of the two-counter machine leading to acceptance, we can deduce at least one representative (actually all representatives except finitely many of them). In particular we see that emptiness of the r.e. set represented by the two-counter machine is equivalent to the emptiness of the state q_0 in our corresponding automaton. $\square$

In Section 4.1, we dismissed pop clauses with equality tests between brothers, that is, clauses of the form $P(f(x_1,\ldots,x_n)) \Leftarrow P_1(x_1),\ldots,P_n(x_n)$ where $x_i = x_j$

for some $i \neq j$. The reason is that it is all too easy to encode intersection clauses using pop clauses with equality tests between brothers, together with standard push clauses; e.g., instead of writing $P(x) \Leftarrow P_1(x), P_2(x)$, we may write the clauses:

$$\begin{aligned} q(f(x,x)) &\Leftarrow P_1(x), P_2(x) \\ P(x) &\Leftarrow q(f(x,y)) \end{aligned}$$

where q is a fresh predicate symbol, and f is any free binary function symbol. It follows:

Proposition 14 *Let $\mathcal{E}$ be any theory, with an associative-commutative symbol $+$ and a free binary symbol f, which is torsion-free w.r.t. four constants. Emptiness is undecidable for standard two-way $\mathcal{E}$-tree automata with equality tests between brothers.*

In the cases of AC, ACU and AG, we can even reduce the number of constants to one, say a, since we may encode the four constants needed earlier as, say, a, $f(a,a)$, $f(a, f(a,a))$ and $f(a, f(a, f(a,a)))$.

We saw in Section 4.2 that general push clauses allowed one to encode intersection clauses, too. The encoding required a unary symbol f to be present in the signature. However we let the reader verify that a similar encoding is possible using a binary AC symbol $+$ in place of the unary symbol f. Let a *general two-way $\mathcal{E}$-tree automaton* be a collection of pop clauses (6), ϵ-clauses (7), and general push clauses (11). The following is then immediate.

Proposition 15 *Let $\mathcal{E}$ be any theory, with an associative-commutative symbol $+$, which is torsion-free w.r.t. four constants. Emptiness is undecidable for general two-way $\mathcal{E}$-tree automata.*

It is interesting to note that unlike in case of theories AC, ACU and AG, intersection-emptiness is decidable for tree automata modulo ACUX, even in the presence of alternation and general two-wayness, and these automata are equally expressive as one-way ACUX automata (Verma, 2004). We finish this enumeration of cases of undecidability by mentioning the following, which deals with the theory A of associativity, without commutativity. This shows that the decidable cases modulo A are even rarer than modulo AC.

Assume our signature contains only one associative symbol $+$ and finitely many constants. Ground terms, e.g., $a+b+a+c+c$ can then be equated with non-empty words, here $abacc$.

Proposition 16 *The languages recognized by one-way A-tree automata on a signature containing only one associative symbol, and finitely many constants, are the context-free languages not containing the empty word.*

One-way A-tree automata are not closed under intersection. Intersection-emptiness is undecidable for one-way A-tree automata. Both results hold even when all free function symbols are constants.

PROOF. Any context-free language L not containing the empty word can be described by a grammar consisting of productions of the form:

$$P \to a \tag{12}$$
$$P \to P_1\ P_2 \tag{13}$$

where P, P_1, P_2 are non-terminals, and a are terminals (letters), and there is a *start non-terminal* P_0. This is the so-called Chomsky normal form (Davis and Weyuker, 1985). The semantics of such productions are described exactly by Horn clauses of the form:

$$\begin{array}{cl} P(a) & \text{represents (12)} \\ P(x+y) \Leftarrow P_1(x), P_2(y) & \text{represents (13)} \end{array}$$

where $+$ is an associative symbol denoting concatenation. The language L is then exactly the set of terms t built on $+$ and the constants a modulo associativity that are recognized at state P_0 in the resulting one-way A-automaton. The converse translation, from one-way A-tree automata to context-free grammars, is obvious.

It is well-known that context-free languages not containing the empty word are not closed under intersection, and that the problem of emptiness of intersection of two context-free languages is undecidable (Davis and Weyuker, 1985), whence the claim. □

Note that emptiness *is* decidable for one-way A-tree automata, even in polynomial time; see Lemma 17 below. Since one-way A-tree automata are not closed under intersection, but they are closed under unions, they are not closed under complementation either.

Alternating one-way A-tree automata are a natural generalization of one-way A-tree automata which are closed under intersection. This generalization has been studied earlier, in the case where the signature contains only the symbol $+$ and unit 0 besides constants, by Okhotin (2001), under the apt name of conjunctive grammars. Just as for alternating AC-tree automata, membership is decidable for conjunctive grammars, again in polynomial time, while emptiness is undecidable. In the AC case, membership is NP-hard (Verma et al., 2005) already for one-way automata (without alternation).

Note that one-way $\mathcal{E}$-tree automata are always closed under unions, trivially, for every equational theory $\mathcal{E}$: if S_1 and S_2 are two one-way $\mathcal{E}$-tree automata, then for every fresh predicate symbol P, the one-way $\mathcal{E}$-tree automaton $S = S_1 \cup S_2 \cup \{P(x) \Leftarrow P_1(x), P(x) \Leftarrow P_2(x)\}$ is such that $L_P(S) = L_{P_1}(S_1) \cup L_{P_2}(S_2)$.

Note finally that emptiness of one-way $\mathcal{E}$-tree automata is always decidable; this can also be deduced from Ohsaki and Takai (2002), Lemma 2, and the fact that Ohsaki's regular equational tree languages coincide with languages of $\mathcal{E}$-tree automata, when $\mathcal{E}$ is a linear theory.

Lemma 17 *Let $\mathcal{E}$ be an equational theory. For every predicate symbol P, for every one-way $\mathcal{E}$-tree automaton S, the set of ground terms recognized at P in S modulo $\mathcal{E}$ is exactly the set of ground terms s such that $s \approx_{\mathcal{E}} t$ for some ground term t recognized at P in S modulo the empty theory.*

In particular, for every equational theory $\mathcal{E}$, emptiness of one-way $\mathcal{E}$-tree automata is decidable in polynomial time.

PROOF. The first claim is by induction on unit resolution proofs. In one direction, let s be any ground term such that $P(s)$ is derivable by a positive unit resolution proof modulo $\mathcal{E}$ from S, and show that there is a ground term t such that $s \approx_{\mathcal{E}} t$ for some ground term t recognized at P in S modulo the empty theory. The least trivial case is when $P(s)$ has been derived by a pop clause $P(f(x, y)) \Leftarrow P_1(x), P_2(y)$: then $s \approx_{\mathcal{E}} f(s_1, s_2)$ such that we have shorter derivations of $P_1(s_1)$ and $P_2(s_2)$ modulo $\mathcal{E}$; the induction hypothesis gives us two ground terms $t_1 \approx_{\mathcal{E}} s_1$ and $t_2 \approx_{\mathcal{E}} s_2$, and the required term t is $f(t_1, t_2)$.

In the other direction, every term t recognized at P in S modulo the empty theory is also recognized at P in S modulo $\mathcal{E}$. We conclude because, by definition, states P in $\mathcal{E}$-tree automata recognize $\mathcal{E}$-equivalence classes of terms.

The second claim follows from the first and the fact that emptiness is decidable in polynomial time for one-way tree automata (Comon et al., 1997; Gécseg and Steinby, 1997), by standard marking techniques. In a nutshell, given any one-way tree automaton, erase every argument of predicate symbols; i.e., replace every pop clause $P(f(x_1, \ldots, x_n)) \Leftarrow P_1(x_1), \ldots, P_n(x_n)$ by the propositional clause $P \Leftarrow P_1, \ldots, P_n$, and every ϵ-clause $P(x) \Leftarrow P'(x)$ by $P \Leftarrow P'$. Then P is non-empty in the input tree automaton if and only if P is derivable from the translated set of propositional Horn clauses; deciding the latter can be done in polynomial time (Dowling and Gallier, 1984). $\Box$

Before we end this section, let us recall that although emptiness is undecidable for alternating AC-tree automata, some problems, in particular the membership problem, are decidable.

Lemma 18 *Membership is decidable for alternating AC-tree automata.*

PROOF. A naive strategy for deciding whether term t is accepted at state q is as follows. Let the set S of subterms of t be defined inductively as follows: $t \in S$, if $f(t_1, \ldots, t_n) \in S$ for free f then each $t_i \in S$, and if $t_1 + t_2 \approx_{AC} s \in S$ then $t_1 \in S$. Then we apply the automata clauses to obtain more and more derivable facts of the form $p(s)$, $s \in S$, till no such facts can be further obtained. Then we check whether $p(t)$ has already been obtained. $\square$

The careful reader will notice that the proof of Proposition 11 establishes that we can encode any r.e. set using alternating AC-tree automata, in some sense. See in particular Claims 12 and 13. If we were indeed able to encode any r.e. set, this would contradict the above lemma. The explanation of the paradox lies in the fact that the encoding of Proposition 11 is a relation, not a function: each counter machine configuration has infinitely many representations as ground atoms, and in our encoding we derive all but finitely many representatives of each of the required configurations.

Recall that emptiness is undecidable but membership is decidable for conjunctive grammars. We have just shown that this is again the case with alternating AC-tree automata.

In the rest of the paper, we exclude alternation from consideration, and deal with two-way AC-tree automata. However, intersection-emptiness will be interesting to us, mainly because of the results of Goubault-Larrecq et al. (2004).

6 Deciding The Constant-Only Case

Because of the negative results of Section 5, we must restrict the format of clauses. We first consider two-way AC-tree automata, as defined in Section 4, restricted to the constant-only case. The latter means that we consider in this section that the signature Σ consists of p constants $a_1, \ldots, a_p$, and exactly one AC symbol $+$. This might seem like a drastic restriction. However we shall understand that this is where all the difficulties concentrate.

Since all free function symbols are constants a_i, pop clauses (6) are just unit clauses $P(a_i)$, or of the form $P(x + y) \Leftarrow P_1(x), P_2(y)$. Two-way AC-tree automata, in the constant-only case, are then AC^0-automata, as defined in Definitions 19 and 22 below.

In general, we shall use clauses of the following form throughout this section:

$$P(x+y) \Leftarrow P_1(x), P_2(y) \quad (14) \qquad\qquad P(x) \Leftarrow P_1(x+y) \quad (17)$$
$$P(a_i) \quad (15) \qquad\qquad \perp \Leftarrow P_1(x), \ldots, P_k(x) \quad (18)$$
$$P(x) \Leftarrow P_1(x) \quad (16) \qquad\qquad \perp \Leftarrow P(u) \quad (19)$$

where x and y are distinct variables, and u is a closed term. Clauses (14) are $+$-*pop clauses*, (15) *base clauses*, (16) ϵ-*clauses*, (17) (AC-)standard $+$-*push clauses*, (18) *final intersection clauses*, or *query clauses* when $k = 1$, and (19) *test clauses*.

Definition 19 (AC^0-Automaton) *An* AC^0-automaton *is a finite set of $+$-pop clauses (14), of base clauses (15), and of ϵ-clauses (16).*

By standard marking techniques, it is decidable whether any given state P of an AC^0-automaton $\mathcal{A}$ is empty in $\mathcal{A}$.

Things get more complex in the presence of final intersection clauses. First, note that ground terms in the constant-only case are finite linear combinations $\sum_{i=1}^{p} n_i a_i$, with $n_i \in \mathbb{N}$ and $\sum_{i=1}^{p} n_i \geq 1$: equivalently, non-zero p-tuples of natural numbers. Now observe that if we read clauses (14), (15), (16) modulo associativity (A) instead of mod AC, what we get is exactly a context-free grammar: (14) is usually written $P \rightarrow P_1, P_2$, (15) is $P \rightarrow a_i$, (16) is $P \rightarrow P_1$. We have already made this remark in the proof of Proposition 16. We can then state the following reformulation of Parikh's Theorem.

Lemma 20 (Parikh) *Call a set of non-zero p-tuples of natural numbers* AC^0-recognizable *if and only if it is $L_P(\mathcal{A})$ for some AC^0-automaton $\mathcal{A}$, modulo the identification of the ground sum $\sum_{i=1}^{p} n_i a_i$ with the p-tuple $(n_1, \ldots, n_p)$.*

For every AC^0-automaton $\mathcal{A}$, $L_P(\mathcal{A})$ is an effective semilinear set, i.e., it is a semilinear set which is computable from $\mathcal{A}$.

The AC^0-recognizable sets are the semilinear sets of non-zero tuples of integers.

The results of Section 5 imply that sets recognized by AC^0-automata extended with general push clauses or even just intersection clauses are in general *not* semilinear. Nonetheless, any finite intersection of semilinear sets is semilinear, so:

Lemma 21 *The satisfiability of sets of clauses (14), (15), (16), (18), (19) is decidable.*

PROOF. Let $\mathcal{A}$ be all non-test, non-final intersection clauses in the given set S, $\perp \Leftarrow P_1^i(x), \ldots, P_{n_i}^i(x)$ be the final intersection clauses in S, and $\perp \Leftarrow P^j(u_j)$ be the test clauses in S. By Lemma 20 the languages $L_{P_j^i}(\mathcal{A})$ and $L_{P^j}(\mathcal{A})$ are effectively semilinear. Then S is $\mathcal{E}$-unsatisfiable if and only if for some i, $L_{P_1^i}(\mathcal{A}) \cap \ldots \cap L_{P_{n_i}^i}(\mathcal{A}) \neq \emptyset$, or for some j, $u_j \in L_{P^j}(\mathcal{A})$, which is effectively decidable. $\square$

In particular, intersection-emptiness of AC^0-automata, and whether a given tuple is recognized by an AC^0-automaton, are decidable problems.

More generally, Parikh's Theorem implies that AC^0-recognizable languages are effectively closed under intersection, union, complementation, and projection, using the obvious fact that any semilinear set A can be effectively converted to an AC^0-automaton recognizing exactly A.

The results of Verma and Goubault-Larrecq (2005, Section 4) imply that satisfiability is also decidable in the presence of standard $+$-push clauses (17). (The case of *conditional* $+$-push clauses is still open.)

Definition 22 (Two-Way AC^0, Standard Two-Way AC^0) *A* standard two-way AC^0-automaton *is a finite set of $+$-pop clauses (14), of base clauses (15), of ϵ-clauses (16), and of standard $+$-push clauses (17).*

A two-way AC^0-automaton *may additionally contain* conditional $+$-push clauses

$$P(x) \Leftarrow P_1(x+y), P_2(y) \tag{20}$$

Showing that the emptiness of standard two-way AC^0-automata is decidable is easy, using resolution techniques. We let the reader check that input resolution with eager subsumption and splitting terminates. Termination is by Dickson's Lemma, which states that the $\leq$ ordering on $\mathbb{N}^p$ is a well-quasi ordering. The resulting algorithm resembles those based on well-structured transition systems (Finkel and Schnoebelen, 2001), except that splitting is necessary as well. The curious reader may find the details in Appendix A. We do not deal with this in the body of the paper, as we are interested in the more general intersection-emptiness problem, and the latter does not seem to be amenable to resolution techniques.

Intersection-emptiness of standard two-way AC^0-automata is decidable. We recapitulate the main results of Verma and Goubault-Larrecq (2005, Section 4).

Lemma 23 *Given a standard two-way AC^0 automaton $\mathcal{A}$, we can effectively construct a BVASS $\mathcal{V}$, such that for every P in $\mathcal{A}$, $\sum_{i=1}^{p} n_i a_i \in L_P(\mathcal{A})$ iff $P(\nu)$ is derivable from $\mathcal{V}$, where $\nu = (n_1, ..., n_p)$.*

Remark 24 *Given any $\nu' \in (\mathbb{N} \cup \infty)^p$, the set $L_<(\nu')$ of all non-zero vectors $\nu \in \mathbb{N}^p$ such that $\nu < \nu'$ is semilinear. Indeed, letting $\nu' = (n'_1, \ldots, n'_p)$, $(n_1, \ldots, n_p)$ is in $L_<(\nu')$ if and only if*

$$\begin{aligned} & n_1 \leq n'_1 \wedge \ldots \wedge n_p \leq n'_p \\ \wedge & (n_1 < n'_1 \vee \ldots \vee n_p < n'_p) \\ \wedge & \ (n_1 \neq 0 \vee \ldots \vee n_p \neq 0) \end{aligned}$$

where $n \leq \infty$ and $n < \infty$ are abbreviations for the formula true. This is a Presburger formula, hence $L_<(\nu')$ is semilinear by Ginsburg and Spanier (1966)'s Theorem. We also write $L_<(\nu')$ the set of sums $\sum_{i=1}^{p} n_i a_i$ where $(n_1, ..., n_p)$ is in $L_<(\nu')$. This should entail no confusion.

Theorem 25 *There is an effective procedure transforming any standard two-way AC^0-automaton $\mathcal{A}$ into an AC^0-automaton $\mathcal{B}$ such that for every P in $\mathcal{A}$, $L_P(\mathcal{A}) = L_P(\mathcal{B})$.*

This is Verma and Goubault-Larrecq (2005, Theorem 2). This is shown as follows. Let $\mathcal{V}$ be the BVASS equivalent to $\mathcal{A}$ computed using Lemma 23. Let us equate vectors $(n_1, \ldots, n_p)$ with sums $\sum_{i=1}^{p} n_i a_i$. Given any standard $+$-push clause C, say $P(x) \Leftarrow P_1(x + y)$, in $\mathcal{A}$, the set L_C of terms recognized at P in $\mathcal{A}$ using this clause is the set of all vectors ν that are strictly covered, i.e., strictly less than some vector ν_1 recognized at P_1 in $\mathcal{V}$. By the properties of $KM(S)$ (Section 3.3), L_C is the set of all vectors such that $\nu < \nu'$ for some generalized configuration ν' such that $P_1(\nu')$ is the conclusion of some covering derivation. Since there are only finitely many covering derivations, L_C is therefore a finite union of sets of the form $L_<(\nu')$, which are semilinear by Remark 24. Note however that this translation from standard two-way AC^0-automata to AC^0-automata involves a construction similar to the Karp-Miller tree construction for VASS, and hence does not give us any primitive-recursive upper bound on the time and space requirement.

In particular,

Corollary 26 *The set of terms recognized at any state of any standard two-way AC^0-automata is effectively semilinear.*

So the languages of standard two-way AC^0-automata are effectively closed under intersection, union, complementation and projection. Also, by Theorem 25 and Lemma 21:

Corollary 27 *The AC-satisfiability of sets of clauses of the form (14)–(19) is decidable.*

Standard two-way AC^0-automata have been criticized in the past because they can only describe semilinear sets, and as such may be felt to lack expressiveness. While this is arguable, notice that the translation from standard two-way AC^0-automata is far from trivial, and probably requires non-primitive recursive time and space. Our feeling is that they describe "non-trivially-semilinear sets", in a similar way as, say, any non-primitive recursive decision problem (i.e., where the answer is a Boolean value) describes "non-trivial Booleans".

Another answer to this critique is that conditional $+$-push clauses, which we cannot handle at the moment, extend this expressive power dramatically. In a conditional $+$-push clause $P(x) \Leftarrow P_1(x + y), Q_1(y), \ldots, Q_n(y)$, the atoms $Q_1(y), \ldots, Q_n(y)$

are called the *conditions*. Let the *condition predicates* be all the symbols Q such that $Q(y)$ is a condition in some clause of $\mathcal{A}$. Call $\mathcal{A}$ a *Petri two-way AC^0-automaton* if and only if, for every condition predicate Q in $\mathcal{A}$, the only clauses in $\mathcal{A}$ of the form $Q(t) \Leftarrow A_1, \ldots, A_n$ are base clauses $Q(a_i)$.

It is easy to extend Lemma 23 to the case of Petri two-way AC^0-automata: replace clauses $P(x) \Leftarrow P_1(x+y), Q_1(y), \ldots, Q_n(y)$ in $\mathcal{A}$ by all BVASS clauses $P'(x) \Leftarrow P_1(x + \delta_i)$, where i ranges over those indices such that $Q_1(a_i)$, ..., $Q_n(a_i)$ are clauses in $\mathcal{A}$, plus the two families of clauses $P^i(x + (-\delta_i)) \Leftarrow P'(x)$ and $P(x + \delta_i) \Leftarrow P^i(x)$, $1 \leq i \leq p$ ensuring that zero vectors are excluded.

It is then also clear that, up to some coding details related again to the inclusion or exclusion of zero vectors, every language accepted by a BVASS is accepted by a Petri two-way AC^0-automaton (even without standard $+$-push clauses). This relies on the expressiveness of conditional $+$-push clauses. Since every language accepted by some VASS is trivially accepted by a BVASS, turning to Petri two-way AC^0-automata would extend the expressive power of our automata to include at least languages expressible as sets of reachable Petri net markings. Emptiness of Petri two-way AC^0-automata reduces (is in fact equivalent to) emptiness of languages accepted by BVASS, which is decidable (Verma and Goubault-Larrecq, 2005). However, we are more interested in intersection-emptiness; and intersection-emptiness includes the problem of intersection-emptiness of BVASS, hence of VASS. The latter problem is in turn equivalent to Petri net reachability (Reutenauer, 1993), which is decidable by the rather complex Mayr-Kosaraju algorithm, and is EXPSPACE-hard (Lipton, 1976). Intersection-emptiness, or equivalently reachability for *branching* VASS is not known to be decidable at the moment. In other words, intersection-emptiness of Petri two-way AC^0-automata is not known to be decidable. *A fortiori* intersection-emptiness of two-way AC^0-automata is not known to be decidable. We would find it surprising nonetheless if any of these problems were undecidable.

Corollary 27 was the main result of this section. We shall use it to show that intersection-emptiness of two-way, non-alternating AC-automata with only standard $+$-push clauses is decidable in Section 8.

7 Closure Properties of One-Way and Standard Two-Way AC-tree automata

Our aim in this section is to show that one-way AC-tree automata are closed under intersection, and that AC-standard two-way AC-tree automata can be effectively converted to one-way AC-tree automata. This way, intersection-emptiness reduces to emptiness of one-way AC-tree automata, which is decidable by Lemma 17.

We first show that two-way AC-tree automata with p AC symbols $+_1, \ldots, +_p$ are equally expressive as two-way AC-tree automata with just one AC symbol $+$, so that we need to study the decidability and closure properties of AC-standard two-way AC-tree automata with only one AC symbol, which is where the main technical challenges lie.

Definition 28 (Standard Translation) *Let Σ' be the signature consisting of all free symbols of Σ, one AC symbol $+$, and $2p$ fresh unary free symbols $\smile_i$ and $\frown_i$, $1 \leq i \leq p$.*

*For every function symbol g, a g-*term *is any term of the form $g(t_1, \ldots, t_n)$. This also makes sense modulo AC, i.e., any term equal to some g-term modulo AC is a g-term; and given any term t, there is a unique function symbol g such that t is a g-term.*

Given any ground term t on the signature Σ, define the ground term t^* on the signature Σ' by:

- *if $t = f(t_1, \ldots, t_n)$ with f a free symbol, then $t^* = f(t_1^*, \ldots, t_n^*)$;*
- *if $t \approx_{\mathrm{AC}} t_1 +_i \ldots +_i t_n$, with $1 \leq i \leq p$, $n \geq 2$, and t_j are not $+_i$-terms, $1 \leq j \leq n$, then $t^* = \frown_i (\smile_i (t_1^*) + \ldots + \smile_i (t_n^*))$.*

The well-parenthesized terms on the signature Σ' are the terms of type U in the following typing system, whose types are the constants U, $U_1, \ldots, U_p$:

$$\frac{t_1 : U \ldots t_n : U}{f(t_1, \ldots, t_n) : U}\ f \text{ free in } \Sigma \qquad \frac{t : U_i \quad t' : U_i}{t + t' : U_i} \qquad \frac{t : U}{\smile_i (t) : U_i} \qquad \frac{t : U_i}{\frown_i (t) : U}$$

for all i, $1 \leq i \leq p$.

Clearly t^* is well-parenthesized for every term t on the signature Σ. Also, the type of every ground term on the signature Σ' is unique if it exists. It follows that the following definition makes sense:

Definition 29 *For every well-typed term in the system of Definition 28 (on the signature Σ'), define t° by:*

- *if f is a free symbol in Σ, and $t = f(t_1, \ldots, t_n)$, then $t^\circ = f(t_1^\circ, \ldots, t_n^\circ)$;*
- *if $t = \smile_i (u)$ or $t = \frown_i (u)$, then $t^\circ = u^\circ$;*
- *if t is a sum $t_1 + \ldots + t_n$, of type U_i, where $t_1, \ldots, t_n$ are not sums, then $t^\circ = t_1^\circ +_i \ldots +_i t_n^\circ$.*

Clearly $(t^*)^\circ \approx_{\mathrm{AC}} t$ for every ground term t on Σ.

Lemma 30 *For any two-way AC-tree automaton $\mathcal{A}$, we can effectively compute a two-way AC-tree automaton $\mathcal{A}^*$ such that $L_P(\mathcal{A}^*)$ is the set of all well-parenthesized ground terms u on the signature Σ' such that $u^\circ \in L_P(\mathcal{A})$, for every predicate symbol P occurring in $\mathcal{A}$. If in addition $\mathcal{A}$ is AC-standard then $\mathcal{A}^*$ is also AC-standard.*

PROOF. For each predicate symbol P occurring in $\mathcal{A}$, create p fresh predicate symbols $P^1, \ldots, P^p$, and add the clauses

$$P^i(\smile_i (x)) \Leftarrow P(x) \tag{21}$$
$$P(\frown_i (x)) \Leftarrow P^i(x) \tag{22}$$
$$P(x) \Leftarrow P^i(\smile_i (x)) \tag{23}$$
$$P^i(x) \Leftarrow P(\frown_i (x)) \tag{24}$$

for every i, $1 \leq i \leq p$, and every predicate symbol P. Call a $+_i$*-clause* any clause whose sole function symbol is $+_i$. Then, in every $+_i$-clause of $\mathcal{A}$, $1 \leq i \leq p$, replace every predicate symbol P by P^i and every occurrence of $+_i$ by $+$. For example, replace the $+_i$-pop clause $P(x +_i y) \Leftarrow P_1(x), P_2(y)$ by the clauses

$$P^i(x + y) \Leftarrow P_1^i(x), P_2^i(y) \tag{25}$$

Let $\mathcal{A}^*$ be the two-way AC-tree automaton thus obtained.

We first claim that every ground term recognized at some state P in $\mathcal{A}^*$ is of type U, and every ground term recognized at some state P^i is of type U_i. In particular, $L_P(\mathcal{A}^*)$ is a set of well-parenthesized ground terms.

We then show by structural induction on the derivation of $P(u)$, resp. $P^i(u)$, from $\mathcal{A}^*$ that $P(u^\circ)$ is derivable from $\mathcal{A}$, for any ground term $u : U$,resp. $u : U_i$. This is straightforward.

Finally we show by structural induction on the derivation of $P(t)$ from $\mathcal{A}$, where t is any ground term on Σ, that $P(u)$ is derivable from $\mathcal{A}^*$ for every $u : U$ such that $u^\circ \approx_{\text{AC}} t$, and $P^i(u)$ is derivable from $\mathcal{A}^*$ for every $u : U_i$ such that $u^\circ \approx_{\text{AC}} t$, $1 \leq i \leq p$. This is again straightforward. $\Box$

The automaton $\mathcal{A}^*$ is *well-parenthesized* in the sense that its state set can be partitioned into $p + 1$ sets $\mathcal{P}$, $\mathcal{P}_1, \ldots, \mathcal{P}_n$ (namely $\mathcal{P}$ is the set of states of $\mathcal{A}$, and $\mathcal{P}_i$ is the set of states of the form P^i, $1 \leq i \leq p$), so that every ground term recognized at some $P \in \mathcal{P}$ is of type U, and every ground term recognized at some $P^i \in \mathcal{P}$ is of type U_i, $1 \leq i \leq p$.

Conversely, we have:

Lemma 31 *Let $\mathcal{B}$ be any one-way AC-tree automaton that accepts only well-typed terms. Then $\mathcal{B}$ can be effectively converted into a one-way AC-tree automaton $\mathcal{B}^\circ$ such that, for every $P \in \mathcal{P}$, $L_P(\mathcal{B}^\circ)$ is the set of all terms u° where u ranges over $L_P(\mathcal{B})$.*

PROOF. As for ordinary (i.e. one-way non-equational) tree automata, it is easy to decide whether some state is redundant, i.e. not involved in any derivation leading to the final state. Hence without loss of generality we may assume that $\mathcal{B}$ contains no redundant state. Since $\mathcal{B}$ accepts only well-typed terms, typing imposes that $\mathcal{B}$ is well-parenthesized and the only pop and ϵ-clauses in $\mathcal{B}$ are of the form:

(1) $P(f(x_1, \ldots, x_n)) \Leftarrow P_1(x_1), \ldots, P_n(x_n)$ with f free, and $P, P_1, \ldots, P_n \in \mathcal{P}$;
(2) or $P^i(x + y) \Leftarrow P_1^i(x), P_2^i(y)$ with $P^i, P_1^i, P_2^i \in \mathcal{P}_i$ $(1 \leq i \leq p)$;
(3) or $P^i(\smile_i (x)) \Leftarrow P(x)$ with $P \in \mathcal{P}$ and $P^i \in \mathcal{P}_i$ $(1 \leq i \leq p)$;
(4) or $P(\frown_i (x)) \Leftarrow P^i(x)$ with $P \in \mathcal{P}$ and $P^i \in \mathcal{P}_i$ $(1 \leq i \leq p)$;
(5) or $P(x) \Leftarrow Q(x)$ with $P, Q \in \mathcal{P}$ or $P^i(x) \Leftarrow Q^i(x)$ with $P^i, Q^i \in \mathcal{P}_i$ $(1 \leq i \leq p)$.

Then replace the clauses of the second kind by $P^i(x +_i y) \Leftarrow P_1^i(x), P_2^i(y)$, clauses of the third kind by $P^i(x) \Leftarrow P(x)$, and clauses of the fourth kind by $P(x) \Leftarrow P^i(x)$. The claim is then clear. $\Box$

In the following, and unless told otherwise, we shall therefore assume that the signature Σ' consists of one AC symbol $+$, the others being free symbols.

Definition 32 (Functional Term, $+$-Part) *A term is* functional *if and only if it is of the form $f(t_1, \ldots, t_n)$, where f is a free function symbol in Σ'.*

*Given any AC-standard two-way AC-tree automaton $\mathcal{A}$, the $+$-*part *$\mathcal{A}_+$ of $\mathcal{A}$ is the subset of all clauses in $\mathcal{A}$ that are either ϵ-clauses (16) $P(x) \Leftarrow P_1(x)$, or $+$-pop clauses (14) $P(x + y) \Leftarrow P_1(x), P_2(y)$, or standard $+$-push clauses (17) $P(x) \Leftarrow P_1(x + y)$.*

7.2 Reusing Derivations

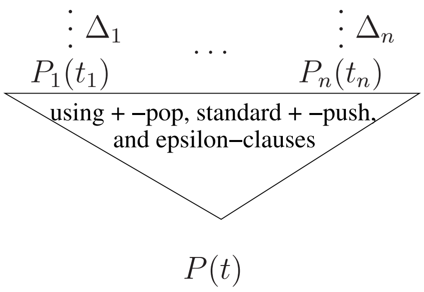
For short, let us call *derivation* of A from a set of definite clauses any positive unit resolution derivation of A from the same set.

Starting from a one-way AC-tree automaton, we first observe that we may slice any derivation in layers, some of them using $+$-pop clauses, the others using pop clauses on free function symbols. The point of Lemma 34 below is that we may freely exchange sublayers for others.

Definition 33 (Functional Support) *Let $\mathcal{A}$ be any two-way AC-tree automaton on Σ, and Δ any derivation of $P(t)$ from $\mathcal{A}$. Let $\Delta_1, \ldots, \Delta_n$ be the set of maximal subderivations of Δ ending with instances of free pop clauses. For $1 \leq j \leq n$, let the conclusion of Δ_j be $P_j(t_j)$, so that t_j is a functional term, $1 \leq j \leq n$. Call the multiset of atoms $P_1(t_1), \ldots, P_n(t_n)$ obtained this way the* functional support *of Δ.*

Let us clarify that in the above definition, two subderivations at two distinct positions are considered distinct, even if they have identical structure. We intend to use the above derivation in cases where Δ involves only clauses of one-way automata. However $\mathcal{A}$ may contain other clauses in general. Note that, going up in Δ from the conclusion $P(t)$, we must eventually encounter an instance of a free pop clause. (In fact, we must eventually encounter an instance of a free pop clause of the form $P(a)$ for some constant a.) Then Δ can be described as in Lemma 34 below.

Lemma 34 *Let Δ be a derivation of the form*



from the set $\mathcal{A}$ of definite clauses, where we mean that $P_j(t_j)$ is the conclusion of Δ_j, $1 \leq j \leq n$, and $P(t)$ is derived from $P_1(t_1)$, ..., $P_n(t_n)$ and only $+$-pop clauses, $+$-push clauses and ϵ-clauses, for some fixed i, $1 \leq i \leq p$.

If t_1, ..., t_n are functional terms, then there are indices $1 \leq i_1 < \ldots < i_k \leq n$ such that:

(1) $t \approx_{\mathrm{AC}} t_{i_1} + \ldots + t_{i_k}$
(2) $\mathcal{A}_+ \models_{\mathrm{AC}} P(x_{i_1} + \ldots + x_{i_k}) \Leftarrow P_1(x_1), \ldots, P_n(x_n)$.
(3) If no $+$-push clause (17) is in $\mathcal{A}$, then $k = n$, i.e., $t \approx_{\mathrm{AC}} t_1 + \ldots + t_n$.

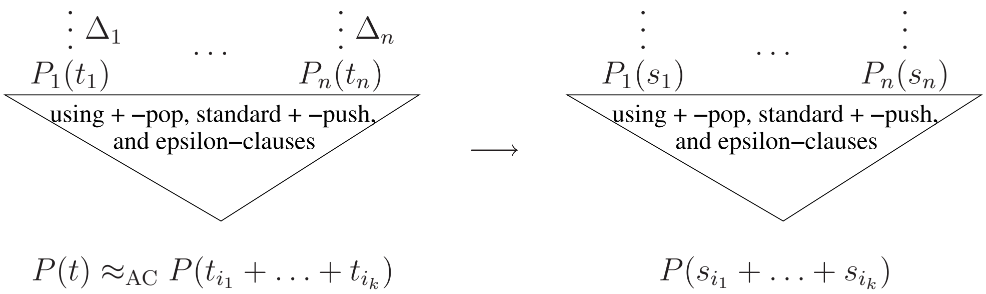
PROOF. The triangle part of the derivation can just sum terms, or extract summands. The point of the Lemma is that, whatever we do, each subscript i will occur at most once in the final sum $t_{i_1} + \ldots + t_{i_k}$. This is because $+$-pop clauses are constrained to add sums coming from disjoint subderivations Δ_i.

We then observe the following

Lemma 35 *Let $\mathcal{A}$ be any set of definite clauses. If $\mathcal{A}_+ \models_{\mathrm{AC}} P(x_{i_1} + \ldots + x_{i_k}) \Leftarrow P_1(x_1), \ldots, P_n(x_n)$, and $P_1(s_1)$, …, $P_n(s_n)$ are ground atoms derivable from $\mathcal{A}$, then $P(s_{i_1} + \ldots + s_{i_k})$ is derivable from $\mathcal{A}$.*

PROOF. From the assumptions we get $\mathcal{A} \models_{\mathrm{AC}} P(s_{i_1} + \ldots + s_{i_k})$, so $\mathcal{A}$ union $\bot \Leftarrow P(s_{i_1} + \ldots + s_{i_k})$ is AC-unsatisfiable by Lemma 1. Since positive unit resolution is complete, the empty clause $\bot$ is then derivable from $\mathcal{A}$ union $\bot \Leftarrow P(s_{i_1} + \ldots + s_{i_k})$. The last step must be a resolution step of $\bot \Leftarrow P(s_{i_1} + \ldots + s_{i_k})$ against some unit clause, which must therefore be $P(s_{i_1} + \ldots + s_{i_k})$. $\square$

Combined with Lemma 34, this will allow us to replace derivations from the two-way AC-tree automaton $\mathcal{A}$ as on the left below by derivations as on the right:



7.3 Intersection of One-Way AC-Tree Automata

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two one-way AC-tree automata built over sets of predicates $\mathcal{P}_1$ and $\mathcal{P}_2$. We will construct a one-way AC-tree automaton $\mathcal{A}$ such that $L_{(P_1,P_2)}(\mathcal{A}) = L_{P_1}(\mathcal{A}_1) \cap L_{P_2}(\mathcal{A}_2)$ for every pair of states P_1 in $\mathcal{P}_1$, P_2 in $\mathcal{P}_2$. Here (P_1, P_2) will be a fresh state, in such a way that there is a one-to-one correspondence between fresh states (P_1, P_2) and pairs of states in $\mathcal{P}_1 \times \mathcal{P}_2$. This should remind the reader of the product construction in ordinary, non-equational, one-way tree automata (Gécseg and Steinby, 1997).

We however need more states, and introduce yet new predicate symbols $\widehat{(P_1, P_2)}$, for all pairs $P_1 \in \mathcal{P}_1$, $P_2 \in \mathcal{P}_2$. We intend the state (P_1, P_2) in the AC-tree automaton $\mathcal{A}_1 \times \mathcal{A}_2$ to be constructed below to recognize the intersection of the languages recognized by states P_1 and P_2 in automata $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. The state $\widehat{(P_1, P_2)}$ is intended to recognize only the functional terms recognized at (P_1, P_2).

We reduce the problem to the constant-only case as follows. Introduce the set $A = \{a_{P_1,P_2} \mid P_1 \in \mathcal{P}_1, P_2 \in \mathcal{P}_2\}$; the constants a_{P_1,P_2} are pairwise distinct and fresh.

In the construction below we use the constant a_{P_1,P_2} as an abstraction for the terms to be recognized at $\widehat{(P_1, P_2)}$.

Define the one-way AC^0-automaton $\mathcal{B}_1 = \mathcal{A}_{1+} \cup \{P_1(a_{P_1,P_2}) \mid P_1 \in \mathcal{P}_1, P_2 \in \mathcal{P}_2\}$. Similarly, define the one-way AC^0-automaton $\mathcal{B}_2 = \mathcal{A}_{2+} \cup \{P_2(a_{P_1,P_2}) \mid P_1 \in \mathcal{P}_1, P_2 \in \mathcal{P}_2\}$. The one-way AC^0-automata $\mathcal{B}_1$ and $\mathcal{B}_2$ are built on the signature $A \cup \{+\}$. For $P_1 \in \mathcal{P}_1$, $P_2 \in \mathcal{P}_2$, $L_{P_1}(\mathcal{B}_1)$ and $L_{P_2}(\mathcal{B}_2)$ are effectively semilinear sets by Lemma 20. So $L_{P_1}(\mathcal{B}_1) \cap L_{P_2}(\mathcal{B}_2)$ is also effectively semilinear. Hence we can define an AC^0-automaton $\mathcal{A}_{P_1,P_2}$ on the signature $A \cup \{+\}$, with a final state F_{P_1,P_2} such that $L_{F_{P_1,P_2}}(\mathcal{A}_{P_1,P_2}) = L_{P_1}(\mathcal{B}_1) \cap L_{P_2}(\mathcal{B}_2)$. We may also assume without loss of generality that the AC^0-automata $\mathcal{A}_{P_1,P_2}$'s are built from mutually disjoint sets of fresh states.

The required one-way AC-tree automaton $\mathcal{A}_1 \times \mathcal{A}_2$ consists of:

(1) a clause $(P_1, P_2)(x) \Leftarrow F_{P_1,P_2}(x)$ for each $P_1 \in \mathcal{P}_1$, $P_2 \in \mathcal{P}_2$;
(2) all clauses of $(\mathcal{A}_{P_1,P_2})_+$, for all $P_1 \in \mathcal{P}_1$, $P_2 \in \mathcal{P}_2$;
(3) a clause $R(x) \Leftarrow \widehat{(P_1', P_2')}(x)$ for each base clause $R(a_{P_1',P_2'})$ in $\mathcal{A}_{P_1,P_2}$, for each $P_1 \in \mathcal{P}_1$, $P_2 \in \mathcal{P}_2$;
(4) a clause $\widehat{(P_1, P_2)}(f(x_1, \ldots, x_n)) \Leftarrow (P_{11}, P_{21})(x_1), \ldots, (P_{1n}, P_{2n})(x_n)$ for each pair of clauses

$$P_1(f(x_1, \ldots, x_n)) \Leftarrow P_{11}(x_1), \ldots, P_{1n}(x_n) \qquad \text{in } \mathcal{A}_1$$
$$P_2(f(x_1, \ldots, x_n)) \Leftarrow P_{21}(x_1), \ldots, P_{2n}(x_n) \qquad \text{in } \mathcal{A}_2$$

where f is free.

Proposition 36 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two one-way AC-tree automata. If $P_1(t)$ and $P_2(t)$ are derivable from $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively, then $(P_1, P_2)(t)$ is derivable from $\mathcal{A}_1 \times \mathcal{A}_2$.*

PROOF. We do induction of the sum of the sizes of the derivations of $P_1(t)$ and $P_2(t)$. Let $t \approx_{\text{AC}} t_1 + \ldots + t_n$ where each t_i is functional. For each $j \in \{1, 2\}$, the derivation of $P_j(t)$ has a functional support of the form $P_{j1}(t_1), \ldots, P_{jn}(t_n)$ by Lemma 34, and $\mathcal{A}_{j+} \models_{\text{AC}} P_j(x_1 + \ldots + x_n) \Leftarrow P_{j1}(x_1), \ldots, P_{jn}(x_n)$.

The atoms $P_{j1}(a_{P_{11},P_{21}}), \ldots, P_{jn}(a_{P_{1n},P_{2n}})$ are derivable in $\mathcal{B}_j$, by construction of $\mathcal{B}_j$. Also, since $\mathcal{A}_{j+} = \mathcal{B}_{j+}$, we observe that $\mathcal{B}_{j+} \models_{\text{AC}} P_j(x_1 + \ldots + x_n) \Leftarrow P_{j1}(x_1), \ldots, P_{jn}(x_n)$. So the atom $P_j(a_{P_{11},P_{21}} + \ldots + a_{P_{1n},P_{2n}})$ is derivable from $\mathcal{B}_j$ by Lemma 35, $j \in \{1, 2\}$. It follows that $F_{P_1,P_2}(a_{P_{11},P_{21}} + \ldots + a_{P_{1n},P_{2n}})$ is derivable from $\mathcal{A}_{P_1,P_2}$.

This derivation has a functional support of the form $R_1(a_{P_{11},P_{21}}), \ldots, R_n(a_{P_{1n},P_{2n}})$, such that $(\mathcal{A}_{P_1,P_2})_+ \models_{\text{AC}} F_{P_1,P_2}(x_1 + \ldots + x_n) \Leftarrow R_1(x_1), \ldots, R_n(x_n)$, by Lemma 34 again. Since $(\mathcal{A}_{P_1,P_2})_+ \subseteq (\mathcal{A}_1 \times \mathcal{A}_2)_+$ by item 2 of the product construction above, it obtains

$$(\mathcal{A}_1 \times \mathcal{A}_2)_+ \models_{\mathrm{AC}} F_{P_1,P_2}(x_1 + \ldots + x_n) \Leftarrow R_1(x_1), \ldots, R_n(x_n) \qquad (*)$$

For $1 \leq i \leq n$ since t_i is functional we have some free f_i of arity k_i and terms $t_i^1, \ldots, t_i^{k_i}$ such that $t_i \approx_{\mathrm{AC}} f_i(t_i^1, \ldots, t_i^{k_i})$. Since $P_{1i}(t_i)$ and $P_{2i}(t_i)$ are in the functional supports of the derivations of $P_1(t)$ and $P_2(t)$ respectively, there are free pop clauses

$$P_{ji}(f_i(x_1, \ldots, x_{k_i})) \Leftarrow P_{ji}^1(x_1), \ldots, P_{ji}^{k_i}(x_{k_i})$$

in $\mathcal{A}_j$, $1 \leq i \leq n$, $j \in \{1,2\}$, and such that for all k, $1 \leq k \leq k_i$, the atoms $P_{ji}^k(t_i^k)$ are derivable from $\mathcal{A}_j$, with derivations strictly smaller than those of $P_j(t)$. By induction hypothesis $(P_{1i}^k, P_{2i}^k)(t_i^k)$ is derivable from $\mathcal{A}_1 \times \mathcal{A}_2$.

By item 4 of the product construction, the clause $(\widehat{P_{1i}, P_{2i}})(f_i(x_1, \ldots, x_{k_i})) \Leftarrow (P_{1i}^1, P_{2i}^1)(x_1), \ldots, (P_{1i}^{k_i}, P_{2i}^{k_i})(x_{k_i})$ is in $\mathcal{A}_1 \times \mathcal{A}_2$. Hence the atom $(\widehat{P_{1i}, P_{2i}})(t_i)$ is derivable from $\mathcal{A}_1 \times \mathcal{A}_2$. Also, since the clause $R_i(a_{P_{1i},P_{2i}})$ is in $\mathcal{A}_{P_1,P_2}$, by item 3 of the product construction the clause $R_i(x) \Leftarrow (\widehat{P_{1i}, P_{2i}})(x)$ is in $\mathcal{A}_1 \times \mathcal{A}_2$. Hence the atom $R_i(t_i)$ is derivable from $\mathcal{A}_1 \times \mathcal{A}_2$, $1 \leq i \leq n$. From (*), and using Lemma 35, $F_{P_1,P_2}(t_1 + \ldots + t_n)$, i.e., $F_{P_1,P_2}(t)$, is derivable from $\mathcal{A}_1 \times \mathcal{A}_2$. Finally we use the clause $(P_1, P_2)(x) \Leftarrow F_{P_1,P_2}(x)$ given by item 1 of the product construction to get a derivation of $(P_1, P_2)(t)$ from $\mathcal{A}_1 \times \mathcal{A}_2$. $\square$

Proposition 37 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two one-way AC-tree automata. For any $P_1 \in \mathcal{P}_1$, $P_2 \in \mathcal{P}_2$, for any ground term t on Σ', if $(P_1, P_2)(t)$ is derivable from $\mathcal{A}_1 \times \mathcal{A}_2$, then $P_1(t)$ and $P_2(t)$ are derivable from $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively.*

PROOF. By inspection of the clauses in $\mathcal{A}_1 \times \mathcal{A}_2$, the only ground terms recognized at predicates of the form $(\widehat{Q_1, Q_2})$ are functional terms, using clauses of item 4 of the product construction. It also follows that for any predicate R such that $R(a_{P_1',P_2'})$ is a base clause in $\mathcal{A}_{P_1,P_2}$, R recognizes only functional terms in $\mathcal{A}_1 \times \mathcal{A}_2$.

We do induction on the size of the derivation of $(P_1, P_2)(t)$. Since $(P_1, P_2)(x) \Leftarrow F_{P_1,P_2}(x)$ is the only clause in $\mathcal{A}_1 \times \mathcal{A}_2$ with the predicate (P_1, P_2) on the left of $\Leftarrow$, the given derivation of $(P_1, P_2)(t)$ ends by an application of $(P_1, P_2)(x) \Leftarrow F_{P_1,P_2}(x)$, so $F_{P_1,P_2}(t)$ is derivable from $\mathcal{A}_1 \times \mathcal{A}_2$ using a strictly smaller derivation. Again from examination of the clauses in $\mathcal{A}_1 \times \mathcal{A}_2$, the derivation of $F_{P_1,P_2}(t)$ has a functional support of the form $(\widehat{P_{11}, P_{21}})(t_1), \ldots, (\widehat{P_{1n}, P_{2n}})(t_n)$, with $t \approx_{\mathrm{AC}} t_1 + \ldots + t_n$, where the predicates $(\widehat{P_{1i}, P_{2i}})$ only recognize functional terms, by the remark above. Furthermore, there are clauses $R_i(x) \Leftarrow (\widehat{P_{1i}, P_{2i}})(x)$ from item 3 of the product construction, so that $R_1(t_1), \ldots, R_n(t_n)$ are derived just below $(\widehat{P_{11}, P_{21}})(t_1), \ldots, (\widehat{P_{1n}, P_{2n}})(t_n)$ respectively.

By item 3 of the product construction again, there are base clauses $R_i(a_{P_{1i},P_{2i}})$ in $\mathcal{A}_{P_1,P_2}$. Also $\mathcal{A}_{P_1,P_2+} \models_{\mathrm{AC}} F_{P_1,P_2}(x_1 + \ldots + x_n) \Leftarrow R_1(x_1), \ldots, R_n(x_n)$, by Lemma 34. So $F_{P_1,P_2}(a_{P_{11},P_{21}} + \ldots + a_{P_{1n},P_{2n}})$ is derivable from $\mathcal{A}_{P_1,P_2}$, by Lemma 35.

For $1 \leq i \leq n$, since t_i is functional, we have some free f_i of arity k_i and terms $t_i^1, \ldots, t_i^{k_i}$ such that $t_i = f_i(t_i^1, \ldots, t_i^{k_i})$. Since $(\widehat{P_{1i}, P_{2i}})(t_i)$ is derivable from $\mathcal{A}_1 \times \mathcal{A}_2$, this must be derived using some clause given by item 4 of the product construction, say $(\widehat{P_{1i}, P_{2i}})(f_i(x_1, \ldots, x_{k_i})) \Leftarrow (P_{1i}^1, P_{2i}^1)(x_1), \ldots, (P_{1i}^{k_i}, P_{2i}^{k_i})(x_{k_i})$ in $\mathcal{A}_1 \times \mathcal{A}_2$. In particular, there are clauses $P_{ji}(f_i(x_1, \ldots, x_{k_i})) \Leftarrow P_{ji}^1(x_1), \ldots, P_{ji}^{k_i}(x_{k_i})$ in $\mathcal{A}_j$, $j \in \{1, 2\}$. Furthermore, for all k, $1 \leq k \leq k_i$, the atom $(P_{1i}^k, P_{2i}^k)(t_i^k)$ is derivable from $\mathcal{A}_1 \times \mathcal{A}_2$ using a derivation strictly smaller than the one of $(P_1, P_2)(t)$. By induction hypothesis $P_{ji}^k(t_i^k)$ is derivable from $\mathcal{A}_j$. It follows that $P_{ji}(t_i)$ is derivable from $\mathcal{A}_j$, $1 \leq i \leq n$, $j \in \{1, 2\}$.

Since $F_{P_1,P_2}(a_{P_{11},P_{21}} + \ldots + a_{P_{1n},P_{2n}})$ is derivable from $\mathcal{A}_{P_1,P_2}$, we obtain that $P_j(a_{P_{11},P_{21}} + \ldots + a_{P_{1n},P_{2n}})$ is derivable from $\mathcal{B}_j$, $j \in \{1, 2\}$, since $L_{F_{P_1,P_2}}(\mathcal{A}_{P_1,P_2}) = L_{P_1}(\mathcal{B}_1) \cap L_{P_2}(\mathcal{B}_2)$. The corresponding derivation must have a functional support $P_{j1}(a_{P_{11},P_{21}}), \ldots, P_{jn}(a_{P_{1n},P_{2n}})$ such that $\mathcal{B}_{j+} \models_{\mathrm{AC}} P_j(x_1 + \ldots + x_n) \Leftarrow P_{j1}(x_1), \ldots, P_{jn}(x_n)$. By definition, $\mathcal{B}_{j+} = \mathcal{A}_{j+}$. Also, since the atoms $P_{j1}(t_1), \ldots, P_{jn}(t_n)$ are derivable from $\mathcal{A}_j$, the atom $P_j(t_1 + \ldots + t_n)$, i.e., $P(t)$, is derivable from $\mathcal{A}_j$. $\square$

If P_1 and P_2 are the chosen final states of $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively then we let (P_1, P_2) be the final state of $\mathcal{A}$. From Proposition 36 and Proposition 37 we have $L_{(P_1,P_2)}(\mathcal{A}_1 \times \mathcal{A}_2) = L_{P_1}(\mathcal{A}_1) \cap L_{P_2}(\mathcal{A}_2)$. We conclude that:

Theorem 38 *The languages recognized by one-way AC-tree automata are effectively closed under intersection.*

By now, several authors have studied one-way AC-tree automata and their variants. In particular, given Lemma 17, and upto details like whether to consider the AC theory or the ACU theory, similar results have been shown by Seidl et al. (2003) and Boneva and Talbot (2005).

7.4 Elimination of Standard $+$-Push Clauses

We now show that adding standard $+$-push clauses does not increase expressiveness of one-way automata. We have already proved this result for the case where all free symbols are constants, i.e. we have shown that two-way AC^0 automata are as expressive as AC^0 automata (Theorem 25). We now consider the general case where we have free symbols of arbitrary arity. As before, we concentrate on the case where there is exactly one AC symbol $+$.

Let $\mathcal{A}$ be an automaton with predicates from $\mathcal{P}$ and containing free pop clauses as well as ϵ-clauses (16), $+$-pop clauses (14), and standard $+$-push clauses (17). We will construct an equivalent automaton $\mathcal{C}$ containing only free pop clauses and ϵ-clauses (16) and $+$-pop clauses (14) (no $+$-push clause (17)). We use the fact that emptiness of a state is decidable for the former class of automata: see Proposition 64 in Appendix A, which also shows that testing emptiness in this class is in NP.

Hence we can assume without loss of generality that $\mathcal{A}$ does not contain any empty state. Introduce a set $A = \{a_P \mid P \in \mathcal{P}\}$ of fresh constants. Define the standard two-way AC^0-automaton $\mathcal{B} = \mathcal{A}_+ \cup \{P(a_P) \mid P \in \mathcal{P}\}$. $\mathcal{B}$ is a standard two-way AC^0 automaton on the signature $A \cup \{+\}$. Hence $L_P(\mathcal{B})$ is a semilinear set for each $P \in \mathcal{P}$, by Corollary 26. Therefore we can construct a one-way AC^0 automaton $\mathcal{A}_P$ with some final state F_P such that $\mathcal{L}_{F_P}(\mathcal{A}_P) = \mathcal{L}_P(\mathcal{B})$. We assume that the $\mathcal{A}_P$'s are based on mutually disjoint sets of fresh predicates.

The required one-way automaton $\mathcal{C}$ consists of

(1) a clause $P(x) \Leftarrow F_P(x)$ for each $P \in \mathcal{P}$;
(2) the clauses of $\mathcal{A}_{P+}$ for each $P \in \mathcal{P}$;
(3) a clause $Q(x) \Leftarrow \widehat{R}(x)$ for each constant clause $Q(a_R)$ in some $\mathcal{A}_P$;
(4) a clause $\widehat{P}(f(x_1, \ldots, x_n)) \Leftarrow P_1(x_1), \ldots, P_n(x_n)$ for each free pop clause $P(f(x_1, \ldots, x_n)) \Leftarrow P_1(x_1), \ldots, P_n(x_n)$ in $\mathcal{A}$;

where $\widehat{P}$ are fresh predicate symbols, for each $P \in \mathcal{P}$.

Lemma 39 *For every ground term t on Σ', if $P(t)$ is derivable in $\mathcal{A}$, then it is derivable in $\mathcal{C}$.*

PROOF. We do induction on the size of the derivation of $P(t)$. Let the derivation of $P(t)$ have functional support $P_1(t_1)$, …, $P_n(t_n)$. From Lemma 34 we have $1 \leq i_1 < \ldots < i_k \leq n$ such that $t \approx_{\mathrm{AC}} t_{i_1} + \ldots + t_{i_k}$ and $\mathcal{A}_+ \models_{\mathrm{AC}} P(x_{i_1} + \ldots + x_{i_k}) \Leftarrow P_1(x_1), \ldots, P_n(x_n)$. Since $\mathcal{A}_+ = \mathcal{B}_+$, we obtain $\mathcal{B}_+ \models_{\mathrm{AC}} P(x_{i_1} + \ldots + x_{i_k}) \Leftarrow P_1(x_1), \ldots, P_n(x_n)$. Also the atoms $P_1(a_{P_1}), \ldots, P_n(a_{P_n})$ are derivable from $\mathcal{B}$, by definition of $\mathcal{B}$. Hence $P(a_{P_{i_1}} + \ldots + a_{P_{i_k}})$ is derivable from $\mathcal{B}$ by Lemma 35. So $F_P(a_{P_{i_1}} + \ldots + a_{P_{i_k}})$ is derivable from $\mathcal{A}_P$. Since $\mathcal{A}_P$ has no clause (17), by Lemma 34 this derivation has a functional support of the form $R_1(a_{P_{i_1}})$, …, $R_k(a_{P_{i_k}})$ and $\mathcal{A}_{P+} \models_{\mathrm{AC}} F_P(x_1 + \ldots + x_k) \Leftarrow R_1(x_1), \ldots, R_k(x_k)$. Since $\mathcal{A}_{P+} \subseteq \mathcal{C}$ by item 2 of the construction, it follows

$$\mathcal{C}_+ \models_{\mathrm{AC}} F_P(x_1 + \ldots + x_k) \Leftarrow R_1(x_1), \ldots, R_k(x_k) \qquad (*)$$

Also since the clause $R_j(a_{P_{i_j}})$ is in $\mathcal{A}_P$, hence $R_j(x) \Leftarrow \widehat{P_{i_j}}(x)$ is in $\mathcal{C}$ for $1 \leq j \leq k$, by item 3 of the construction.

For $1 \leq i \leq n$ since t_i is functional we have some free f_i of arity k_i and terms $t_i^1, \ldots, t_i^{k_i}$ such that $t_i \approx_{\mathrm{AC}} f_i(t_i^1, \ldots, t_i^{k_i})$. Since $P_i(t_i)$ is in the functional support of the derivation of $P(t)$, there is some clause $P_i(f_i(x_1, \ldots, x_{k_i})) \Leftarrow P_i^1(x_1), \ldots,$

$P_i^{k_i}(x_{k_i})$ such that for $1 \leq j \leq k_i$, the atom $P_i^j(t_i^j)$ is derivable from $\mathcal{A}$ using a derivation strictly smaller than that of $P(t)$. By induction hypothesis $P_i^j(t_i^j)$ is derivable from $\mathcal{C}$ for $1 \leq j \leq k_i$. So, for $1 \leq i \leq n$ $\widehat{P_i}(t_i)$ is derivable from $\mathcal{C}$ using the clause $\widehat{P_i}(f_i(x_1, \ldots, x_{k_i})) \Leftarrow P_i^1(x_1), \ldots, P_i^{k_i}(x_{k_i})$ given in item 4 of the construction. Hence for $1 \leq j \leq k$, $R_j(t_{i_j})$ is derivable from $\mathcal{C}$ using the clause $R_j(x) \Leftarrow \widehat{P_{i_j}}(x)$.

Hence from (*) and by Lemma 35, $F_P(t_{i_1} + \ldots + t_{i_k})$, that is, $F_P(t)$, is derivable from $\mathcal{C}$. Finally we use the clause $P(x) \Leftarrow F_P(x)$ from item 1 of the construction to get a derivation of $P(t)$ from $\mathcal{C}$. $\square$

Lemma 40 *For every $P \in \mathcal{P}$, for every ground term t on Σ', if $P(t)$ is derivable from $\mathcal{C}$ then it is derivable from $\mathcal{A}$.*

PROOF. We do induction on the size of the derivation of $P(t)$. Since $P(x) \Leftarrow F_P(x)$ is the only clause with P on the left of $\Leftarrow$, the derivation of $P(t)$ uses the clause $P(x) \Leftarrow F_P(x)$ as the last clause, so $F_P(t)$ is derivable from $\mathcal{C}$ using a derivation strictly smaller than that of $P(t)$. From Lemma 34 and from examination of the clauses in $\mathcal{C}$, the derivation of $F_P(t)$ has a functional support of the form $\widehat{P_1}(t_1), \ldots, \widehat{P_n}(t_n)$ such that $t \approx_{\mathrm{AC}} t_1 + \ldots + t_n$, the clause used immediately above the root of the derivation of $\widehat{P_i}(t_i)$ is of the form $R_i(x) \Leftarrow \widehat{P_i}(x)$ and $\mathcal{A}_{P+} \models_{\mathrm{AC}} F_P(x_1 + \ldots + x_n) \Leftarrow R_1(x_1), \ldots, R_n(x_n)$. Also for $1 \leq i \leq n$, the clause $R_i(a_{P_i})$ is in $\mathcal{A}_P$. Hence $F_P(a_{P_1} + \ldots + a_{P_n})$ is derivable from $\mathcal{A}_P$. So $P(a_{P_1} + \ldots + a_{P_n})$ is derivable from $\mathcal{B}$. By Lemma 34, this derivation has a functional support of the form $P_1(a_{P_1}), \ldots, P_n(a_{P_n}), Q_1(a_{Q_1}), \ldots, Q_m(a_{Q_m})$ $(m \geq 0)$ and $\mathcal{B}_+ \models_{\mathrm{AC}} P(x_1 + \ldots + x_n) \Leftarrow P_1(x_1), \ldots, P_n(x_n), Q_1(y_1), \ldots, Q_m(y_m)$. By definition $\mathcal{B}_+ = \mathcal{A}_+$, hence

$$\mathcal{A}_+ \models_{\mathrm{AC}} P(x_1 + \ldots + x_n) \Leftarrow P_1(x_1), \ldots, P_n(x_n), Q_1(y_1), \ldots, Q_m(y_m) \qquad (*)$$

For $1 \leq i \leq n$, since t_i is functional we have some free f_i of arity k_i and terms $t_i^1, \ldots, t_i^{k_i}$ such that $t_i = f_i(t_i^1, \ldots, t_i^{k_i})$. Since $\widehat{P_i}(t_i)$ is in the functional support of the derivation of $F_P(t)$ hence we have a clause $\widehat{P_i}(f_i(x_1, \ldots, x_{k_i})) \Leftarrow P_i^1(x_1), \ldots, P_i^{k_i}(x_{k_i})$ in $\mathcal{C}$ corresponding to some clause $P_i(f_i(x_1, \ldots, x_{k_i})) \Leftarrow P_i^1(x_1), \ldots, P_i^{k_i}(x_{k_i})$ in $\mathcal{A}$ and for $1 \leq j \leq k_i$, the atom $P_i^j(t_i^j)$ is derivable from $\mathcal{C}$ using a derivation strictly smaller than that of $P(t)$. By induction hypothesis $P_i^j(t_i^j)$ is derivable from $\mathcal{A}$ for $1 \leq j \leq k_i$. Hence for $1 \leq i \leq n$, $P_i(t_i)$ is derivable from $\mathcal{A}$ using the clause $P_i(f_i(x_1, \ldots, x_{k_i})) \Leftarrow P_i^1(x_1), \ldots, P_i^{k_i}(x_{k_i})$. Also since $\mathcal{A}$ contains no empty states, for $1 \leq i \leq m$ we have ground terms s_i such that $Q_i(s_i)$ is derivable from $\mathcal{A}$. So from (*), $P(t_1 + \ldots + t_n)$, i.e., $P(t)$, is derivable from $\mathcal{A}$. $\square$

If P is the final state of $\mathcal{A}$ then we let P be the final state of $\mathcal{C}$. From Lemmas 39 and 40, $L_P(\mathcal{C}) = L_P(\mathcal{A})$. We conclude that

Theorem 41 *Standard two-way AC-tree automata without free push clauses can be effectively reduced to equivalent one-way AC-tree automata.*

7.5 Elimination of Free Push Clauses

We have seen that we can add standard $+$-push clauses to one-way AC-tree automata without increasing their expressiveness. Now we show that we can further add free push clauses without increasing expressiveness, by showing how to eliminate the free push clauses. (Note that the free push clauses need not be standard, contrarily to $+$-push clauses, i.e., we consider AC-standard two-way AC-tree automata.)

We use a saturation procedure that iteratively adds new ϵ-clauses so that finally the free push clauses become redundant.

We first define one step of the saturation procedure. Let $\mathcal{A}$ be an AC-standard two-way AC-tree automaton with predicates from $\mathcal{P}$. Let $\mathcal{A}_1$ be the part of $\mathcal{A}$ without the free push clauses.

Trivially $\mathcal{A}_+ \subseteq \mathcal{A}_1$.

We define the transition relation $\triangleright$ as follows. We let $\mathcal{A} \triangleright \mathcal{A} \cup \{R(x_i) \Leftarrow Q_i(x_i)\}$ provided:

1 $\mathcal{A}$ contains a free push clause $R(x_i) \Leftarrow P(f(x_1, \ldots, x_n)), P_1(x_{i_1}), \ldots, P_k(x_{i_k})$;
2 $\mathcal{A}$ contains a free pop clause $Q(f(x_1, \ldots, x_n)) \Leftarrow Q_1(x_1), \ldots, Q_n(x_n)$ (with the same free function symbol f);
3 $\mathcal{A}_+ \models_{\mathrm{AC}} C$ for some clause $C = P(x) \Leftarrow Q(x), R_1(x_1), \ldots, R_p(x_p)$ $(p \geq 0)$;
4 for each $j \in \{1, \ldots, p\}$ there is a ground term s_j on Σ' such that R_j recognizes s_j in $\mathcal{A}_1$;
5 for each $j \in \{1, \ldots, k\}$ there is a ground term t_{i_j} on Σ' such that both P_i and Q_{i_j} recognize t_{i_j} in $\mathcal{A}_1$;
6 for each $j \in \{1, \ldots, n\} \setminus \{i, i_1, \ldots, i_k\}$, there is a ground term t_j on Σ' such that Q_j recognizes t_j in $\mathcal{A}_1$;
7 and no clause $R(x) \Leftarrow Q_i(x)$ is already in $\mathcal{A}$;

Some remarks are necessary. In step 3, it is sufficient to consider the (finitely many) clauses in which $R_1, \ldots, R_p$ are mutually distinct (the so-called *condensed* clauses). This is because, were R_1 equal to R_2 for example, the clauses $P(x) \Leftarrow Q(x), R_1(x_1), R_2(x_2), \ldots, R_p(x_p)$ and $P(x) \Leftarrow Q(x), R_2(x_2), \ldots, R_p(x_p)$ would be logically equivalent. For each condensed clause $C = P(x) \Leftarrow Q(x), R_1(x_1), \ldots, R_p(x_p)$, the condition $\mathcal{A}_+ \models_{\mathrm{AC}} C$ is then decidable by skolemizing, i.e., by testing whether $\mathcal{A}_+$ union the clauses $-P(a)$, $+Q(a)$, $+R_1(a_1)$, $\ldots$, $R_p(a_p)$ is AC-unsatisfiable, where a, a_1, $\ldots$, a_p are fresh constants. Equivalently, by testing whether $\mathcal{A}_+$ union the final intersection clause $\perp \Leftarrow P'(x), P(x)$ and the unit clauses $+P'(a)$, $+Q(a)$, $+R_1(a_1)$, …, $R_p(a_p)$ is AC-unsatisfiable, where P' is some fresh predicate symbol. This is decidable by Corollary 27, since this clause set is a standard two-way AC^0-automaton.

Also, from Theorems 38 and 41 emptiness and intersection-emptiness problems are decidable for standard two-way AC-tree automata without free push clauses, so conditions 4, 5 and 6 are effectively testable. Hence we can effectively check whether $\mathcal{A} \rhd \mathcal{A} \cup \{R(x_i) \Leftarrow Q_i(x_i)\}$. Since there are only finitely many ϵ-clauses $R(x_i) \Leftarrow Q_i(x_i)$, we can also compute all ϵ-clauses $R(x_i) \Leftarrow Q_i(x_i)$ such that $\mathcal{A} \rhd \mathcal{A} \cup \{R(x_i) \Leftarrow Q_i(x_i)\}$.

This saturation step is harmless:

Lemma 42 *Let $\mathcal{A}$ be any AC-standard two-way AC-tree automaton. If $\mathcal{A} \rhd \mathcal{A} \cup \{R(x_i) \Leftarrow Q_i(x_i)\}$, then $\mathcal{A} \cup \{R(x_i) \Leftarrow Q_i(x_i)\}$ and $\mathcal{A}$ derive exactly the same ground atoms on Σ'.*

PROOF. Every ground atom derivable from $\mathcal{A}$ is clearly derivable from $\mathcal{A} \cup \{R(x_i) \Leftarrow Q_i(x_i)\}$. Conversely, it is sufficient to show that $R(t_i)$ is derivable from $\mathcal{A}$ assuming $Q_i(t_i)$ is derivable from $\mathcal{A}$. For $j \in \{1, \ldots, n\} \setminus \{i\}$, let t_j be as in item 5 (if j is in $\{i_1, \ldots, i_k\}$) or as in item 6 (otherwise) above. $Q(f(t_1, \ldots, t_n))$ is derivable from $\mathcal{A}$ using the free pop clause given in item 2. For $j \in \{1, \ldots, p\}$ let s_j be as in item 4 above. Then $P(f(t_1, \ldots, t_n))$ is derivable from $\mathcal{A}$ using the clause $P(x) \Leftarrow Q(x), R_1(x_1), \ldots, R_p(x_p)$ of item 3, and the facts $Q(f(t_1, \ldots, t_n))$ and $R_1(s_1)$, …, $R_p(s_p)$. $R(t_i)$ is then derivable using the free push clause given in item 1, the fact $P(f(t_1, \ldots, t_n))$ and the facts $P_1(t_{i_1})$, …, $P_k(t_{i_k})$ guaranteed by item 5. $\square$

Given an AC-standard two-way AC-tree automaton $\mathcal{A}$ our saturation procedure consists of (don't care non-deterministically) generating a sequence $\mathcal{A}_0 (= \mathcal{A}) \rhd \mathcal{A}_1 \rhd \mathcal{A}_2$... until no new clause can be added. This always terminates because there are only a finite number of ϵ-clauses $R(x_i) \Leftarrow Q_i(x_i)$ possible. Let the final, saturated, AC-standard two-way AC-tree automaton be $\mathcal{B}$. Then we remove the free push clauses from $\mathcal{B}$ to get $\mathcal{B}_1$. This step is also harmless:

Lemma 43 *Let $\mathcal{B}$ be a AC-standard two-way AC-tree automaton in $\rhd$-normal form, and $\mathcal{B}_1$ be obtained from $\mathcal{B}$ by removing all free push clauses. The set of ground atoms on Σ' derivable from $\mathcal{B}$ and from $\mathcal{B}_1$ are the same.*

PROOF. That any ground atom derivable from $\mathcal{B}_1$ is also derivable from $\mathcal{B}$ is obvious. To show the converse, it is sufficient to show that a derivation from $\mathcal{B}$ which uses a free push clause only in the last step and nowhere else, can be converted to a derivation from $\mathcal{B}_1$; the general case follows by induction on derivations.

Assume we have got a derivation of $R(t_i)$ using the free push clause $R(x_i) \Leftarrow P(f(x_1, \ldots, x_n), P_1(x_{i_1}), \ldots, P_k(x_{i_k})$ in the last step. Hence the atoms $P(f(t_1, \ldots, t_n))$, $P_1(t_{i_1}), \ldots, P_k(t_{i_k})$ are derivable in $\mathcal{B}_1$. From Lemma 34 the derivation of $P(f(t_1, \ldots, t_n))$ has a functional support of the form $Q(f(t_1, \ldots, t_n))$, $R_1(s_1)$, …, $R_p(s_p)$ $(p \geq 0)$ such that $\mathcal{B}_+ \models_{\mathrm{AC}} P(x) \Leftarrow Q(x), R_1(x_1), \ldots, R_p(x_p)$. The derivation of $Q(f(t_1, \ldots, t_n))$ must use some clause $Q(f(x_1, \ldots, x_n)) \Leftarrow Q_1(x_1), \ldots, Q_p(x_p)$ as the last clause. Hence $Q_1(t_1), \ldots, Q_n(t_n)$ are derivable from $\mathcal{B}_1$. Since conditions 1–6 are satisfied, $\mathcal{B} \rhd \mathcal{B} \cup \{R(x_i) \Leftarrow Q_i(x_i)\}$, unless some clause $R(x) \Leftarrow Q_i(x)$ is already in $\mathcal{B}$. But $\mathcal{B}$ is $\rhd$-normal, so $R(x) \Leftarrow Q_i(x)$ is in $\mathcal{B}$, hence in $\mathcal{B}_1$. Using the latter clause, and since $Q_i(t_i)$ is derivable from $\mathcal{B}_1$, we obtain that $R(t_i)$ is derivable from $\mathcal{B}_1$. $\square$

Hence the two-way automaton $\mathcal{A}$ is equivalent to the automaton $\mathcal{B}_1$. Hence free push clauses can be effectively eliminated from an AC-standard two-way AC automaton. From Theorem 41 it follows:

Theorem 44 *AC-standard two-way AC-tree automata can be effectively reduced to one-way AC-tree automata recognizing the same language.*

Corollary 45 *AC-standard two-way AC-tree automata are effectively closed under intersection and their intersection-emptiness problem is decidable.*

Although we have focused mainly on decidability in this paper, recent results (Verma et al., 2005) show that intersection-non-emptiness is NP-complete in the absence of $+$-push clauses, when the number of languages to be intersected is bounded by a fixed constant. The NP-completeness result holds even when the automata are restricted to be one-way AC^0. It follows that the decision problem mentioned in Corollary 45, when restricted to a fixed number of languages to be intersected, is NP-hard. In case the number of languages to be intersected in not bounded, intersection-non-emptiness is DEXPTIME-hard, since intersection-non-emptiness of non-equational tree automata is a special case of it (Seidl, 1994).

8 Going Further

The decision procedure of Section 7 eventually adds new ϵ-clauses, and removes free push clauses. There is another way to derive the same result, using resolution and splittingless splitting. This is based on results by Goubault-Larrecq et al. (2004), which we recapitulate. This will allow us to show, additionally, that intersection-emptiness is decidable for two-way AC-tree automata (not just for AC-standard such automata) as soon as it is for constant-only two-way AC-tree automata. Therefore, the constant-only case indeed concentrates all difficulties, as we have claimed earlier.

Again, we only need to consider the case of one AC symbol $+$, by Lemma 30 and Lemma 31. We require the following definition.

Definition 46 (Blocks, Complex Clauses) *A* block *is any clause of the form* $\pm_1 P_1(x) \vee \ldots \vee \pm_n P_n(x)$*, for the* same *variable* x*. We abbreviate such blocks* $B(x)$*.*

A complex clause *is any clause of the form* $\bigvee_{i=1}^{m} \pm_i P_i(f(x_1, \ldots, x_n)) \vee B_1(x_1) \vee \ldots \vee B_n(x_n)$*, where* $B_1, \ldots, B_n$ *are blocks and* $m \geq 1$ *(with the* same f *and the same set of variables* $x_1, \ldots, x_n$ *as arguments of* f*)*

Note that every clause from any two-way, alternating $\mathcal{E}$-tree automata is either a block or a complex clause.

Imagine we would like to decide intersection-emptiness of two-way AC-tree automata. Let S be a set of two-way AC-tree automata clauses, including query and final intersection clauses. To decide whether S is AC-satisfiable, we use ordered resolution with selection, eager ϵ-splitting, and elimination of tautologies and forward subsumed clauses. As shown in Goubault-Larrecq et al. (2004, Section 4.1), this only derives blocks and complex clauses again, and there are only finitely many of them. So this strategy indeed decides S, in deterministic exponential time... modulo the empty theory. In the case where there is an AC symbol $+$, this strategy in general does not terminate (Goubault-Larrecq et al., 2004, Section 4.2), because the $+$*-clauses*, i.e., the clauses whose only function symbol is $+$, generated by resolution can grow without bounds.

The intuition behind the procedures of Goubault-Larrecq et al. (2004) is as follows. Imagine for the moment that we are using just resolution to decide the AC-satisfiability of automata clauses, modulo AC. Then any proof is a tree whose nodes are labeled by clauses; if resolution is applied to the side premises $C_1, \ldots, C_n$ and the main premise C, with conclusion C', then C' will label a node whose sons are labeled with $C_1, \ldots, C_n, C$. Splitting would complicate matters quite a lot here. If we overlook the problem with splitting for the moment, and if we ignore the necessity of using splitting literals q, Goubault-Larrecq et al. (2004) show that, as long as we deal with blocks and complex clauses with only free function symbols (all but $+$), only finitely many clauses, either blocks or similar complex clauses, can be produced. As soon as $+$ comes into play, we may get larger and larger $+$-clauses. But, if such clauses eventually participate in deriving the empty clause, it must be the case that one $+$-clause thus derived eventually resolves with other clauses to get a conclusion C that is either directly the empty clause, or can resolve with complex clauses not containing $+$. Since no term headed by $+$ unifies with a term headed by f, with $f \neq +$, and provided we only unify on maximal atoms, C can only be a disjunction of literals of the form $\pm P(x)$, with x a variable. Hence C must split into blocks.

This is tentatively pictured in Figure 3; the leaves of the derivation (at the top) are clauses in the initial clause set S. Resolution steps inside the white zones are those among blocks and *free complex clauses*, that is complex clauses in which $+$ does not occur. These must terminate since they only generate finitely many clauses. Resolution steps inside the grey zones produce arbitrarily many, arbitrarily large $+$-clauses. This leads us to the following idea: instead of applying resolution inside the grey zones, try to *guess* the fat dots, which are the interface points between grey zones and white zones. Forbid resolution to act on $+$-clauses (this prevents us from using resolution to derive clauses inside the grey zones), and compensate this by adding a rule that infers the fat dot clauses, at the bottom of grey zones, directly from the clauses at the top of grey zones: this is the *oracle rule*. Although this does not seem practical at all, Goubault-Larrecq et al. (2004) use this to derive a complete (but unsound) oracle, and therefore give a sufficient condition for AC-satisfiability; this was then used to automatically verify the IKA.1 protocol in the so-called pure eavesdropper case. We use this idea to derive decidability results instead. Let us notice however that the fat dots, which are splittable disjunctions of blocks, are only finitely many.

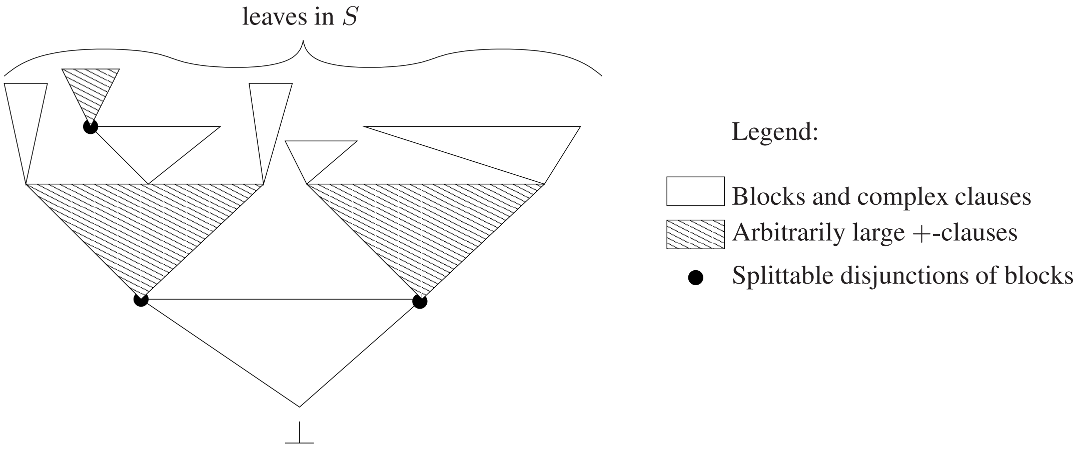


Fig. 3. Grey zones, fat dots, white zones

The results of Goubault-Larrecq et al. (2004) apply to clauses of one of the following form. We take the numbering from op.cit. Also, we let $\mathcal{Q}_0$ be the set of all splitting literals q of the form $\ulcorner B(x) \urcorner$, where $B(x)$ is any non-empty negative block. If $\mathcal{P}$ contains p predicate symbols, then $\mathcal{Q}_0$ contains $2^p - 1$ elements. The notation $[\vee + q]$ denotes an optional literal $+q$ in disjunction with the rest of the clause. A *non-trivial clause* is a clause containing at least one function symbol.

Definition 47 (4,5,6) *Consider the following kinds of Horn clauses.*

(4) $C[\vee + q]$, *where C is a block and $q \in \mathcal{Q}_0$;*
(5) $C \vee -q_1 \vee \ldots \vee -q_m[\vee + q]$, *where C is a free complex clause, $m \geq 0$, and $q_1, \ldots, q_m, q \in \mathcal{Q}_0$;*

(6) $C[\vee + q]$, *where* C *is a non-trivial* $+$*-clause, and* $q \in \mathcal{Q}_0$.

Clearly all clauses that we consider in this paper are of one of the forms **(4)**, **(5)**, **(6)**.

Let us abstract whatever may happen inside the grey zone by a unique rule: starting from a set S of clauses of type **(4)**, **(5)** or **(6)**, we guess which kind of clauses may be the fat dots terminating the grey zones. As noticed in Goubault-Larrecq et al. (2004, Section 4.4), these fat dots are the *candidates*. This is Goubault-Larrecq et al. (2004, Definition 2).

Definition 48 (Candidate, Oracle) *A* candidate *is any Horn clause of the form* $B_1(x_1) \vee \ldots \vee B_n(x_n)[\vee + q]$*, where the* x_i*s are pairwise distinct,* $B_i(x_i)$ *is a non-empty block for every* i*,* $1 \leq i \leq n$*, and* $q \in \mathcal{Q}_0$.

A grey oracle *is any function* $\mathcal{O}$ *mapping every set of clauses of type* **(4)**, **(5)**, *or* **(6)**, *to a set of candidates containing all those deducible by grey resolution.* Grey resolution *is ordered resolution with selection, where at least one premise is of type* **(6)**.

Conversely, call *white resolution* the rule of ordered resolution with selection applied to premises of type **(4)** or **(5)** only.

The following is Goubault-Larrecq et al. (2004, Corollary 3), specialized to the case where Σ_0 consists of all free function symbols, and $\Sigma = \{+\}$, and to the equational theory AC. (This corollary applies to any equational theory E that is *simple*, i.e., such that there is a computable strict, stable ordering $\succ$ closed under context applications and compatible with E, such that $f(x_1, \ldots, x_n) \succ x_i$ for every i, $1 \leq i \leq n$, and every function symbol f. Clearly AC is a simple equational theory.)

Proposition 49 *Let* $\mathcal{O}$ *be any grey oracle. Let* $\succ_{\mathrm{AC}}$ *be any computable strict, stable ordering compatible with* AC *such that* $f(x_1, \ldots, x_n) \succ_{\mathrm{AC}} x_i$ *for every* i*,* $1 \leq i \leq n$*, for every function symbol* f*. Let* sel *be the following selection function:*

- *If* C *is a Horn* $+$*-clause, possibly in disjunction with* $+q$*,* $q \in \mathcal{Q}_0$*, then:*
  - *If* C *contains a negative literal* $-P(t)$ *with* t *not a variable, then let* $\mathsf{sel}\,(C)$ *be* $\{-P(t)\}$*;*
  - *Otherwise, if* C *can be written as* $A \Leftarrow H, P_1(x), \ldots, P_m(x)$ *where* x *is free neither in* A *nor in the body (i.e., conjunction of atoms)* H *and* $m \geq 1$ *(in other words, if there is a variable* x *free on the right of* $\Leftarrow$ *but not on the left), then let* $\mathsf{sel}\,(C)$ *be* $\{-P_1(x), \ldots, -P_m(x)\}$.
  - *Otherwise,* $\mathsf{sel}\,(C)$ *is empty.*
- *If* C *is a clause containing no* $+$ *function symbol, then:*
  - *if* C *contains a negative literal* $-q$ *with* $q \in \mathcal{Q}$*, then* $\mathsf{sel}\,(C) = \{-q\}$*;*
  - *otherwise, let* $\max(C)$ *be the set of maximal literals in* C *for* $\succ$*, then define* $\mathsf{sel}\,(C)$ *as the subset of those negative literals in* $\max(C)$.

For simplicity, say "resolution" for "ordered resolution with selection with ordering $\succ_{\mathrm{AC}}$ and selection function sel *".*

Then white resolution together with the grey oracle rule*: to S add $\mathcal{O}(S)$, is complete for every set of clauses of type* **(4)***,* **(5)***, or* **(6)***. In other words, for every set S_1 of clauses of type* **(4)***,* **(5)***, or* **(6)***, if S_1 is* AC*-unsatisfiable, then the empty clause can be derived from S_1 by white resolution and the grey oracle rule. Moreover, completeness is retained when removing tautologies, forward subsumed clauses, and ϵ-splitting of clauses not of type* **(6)***.*

Proposition 49 requires quite many assumptions. An ordering $\succ_{\mathrm{AC}}$ obeying these assumptions always exists (Goubault-Larrecq et al., 2004, Section 4.4). To be fair, we shall never need to know the detailed definition of either $\succ_{\mathrm{AC}}$ or sel .

Additionally, we observe that white resolution alone terminates with this choice of $\succ_{\mathrm{AC}}$ and sel , while only generating clauses of type **(4)** or **(5)** (Goubault-Larrecq et al., 2004, Section 4.4).

Since there are only finitely many candidates, it follows that AC-satisfiability of sets of clauses of type **(4)**, **(5)**, and **(6)** reduces to finding a *sound*, computable grey oracle $\mathcal{O}$.

Definition 50 *A grey oracle $\mathcal{O}$ is* sound *if and only if, for every set S of clauses of type* **(4)***,* **(5)***, and* **(6)***, $\mathcal{O}(S)$ is a set of candidates that are semantic consequences modulo* AC *of the clauses in S.*

Theorem 51 (Main Theorem) *If there is a sound computable grey oracle, then* AC*-satisfiability of sets of clauses of type* **(4)***,* **(5)***, and* **(6)** *is decidable.*

PROOF. If S_1 is AC-unsatisfiable, by completeness (Proposition 49), we may derive the empty clause from S_1 by white resolution and the grey oracle rule. Conversely, since both white resolution and the grey oracle rule only derive logical consequences of S_1 modulo AC, if we can derive the empty clause, then S_1 is AC-unsatisfiable. □

Since any alternating AC-tree automaton consists of clauses of this form, AC-unsatisfiability of such clause sets is undecidable, so there can be no sound computable grey oracle in general.

In the case of non-alternating automata, we can further restrict the clauses of type **(4)**, **(5)**, **(6)**. This is formalized by the notion of *alternation-free* clauses (Definition 52 below). We shall then relax soundness for oracles so that the oracle $\mathcal{O}$ is only required to be sound on sets of alternation-free clauses. Then we shall show that finding such a sound oracle is equivalent to solving AC-satisfiability in the constant-only case.

Definition 52 (Alternation-Free) *A term t is* linear *if and only if every variable occurs at most once in t. An atom is linear if and only if it is of the form q, $q \in \mathcal{Q}$, or $P(t)$ with t a linear term. We also consider that the symbol $\perp$ is linear.*

A Horn clause $A \Leftarrow A_1, \ldots, A_n$ is alternation-free *if and only if A, $A_1, \ldots, A_n$ are linear, and every variable free in A occurs at most once in $A_1, \ldots, A_n$.*

Note that pop clauses (6), ϵ-clauses (7), push clauses (9) (whether standard or conditional) are alternation-free. Query clauses (see Lemma 2) and even final intersection clauses (Lemma 3) are also alternation-free. On the other hand, intersection clauses (8) are not, and the only general push clauses (11) that are alternation-free are in fact just push clauses.

The point is that resolving alternation-free clauses together only produces alternation-free clauses, under mild assumptions, as we see shortly. We need to observe that unifiers have a special form:

Definition 53 (E, F-Linear) *Let E and F be two sets of variables, and σ be any substitution. We say that σ is E, F-*linear *if and only if:*

(1) for every variable z, $z\sigma$ is a linear term;
(2) for every variable z, there is at most one variable x in E such that z is free in $x\sigma$, and at most one variable y in F such that z is free in $y\sigma$.

We observe that elements of complete sets of unifiers modulo AC, as used in resolution between alternation-free clauses, which unify $s \doteq t$, are E, F-linear, for some well-chosen disjoint sets E and F, in the cases that we are interested in. This is the topic of Lemma 54 and Lemma 55 below.

Lemma 54 *Let s and t be any two linear terms, where only free function symbols occur, and with disjoint sets of free variables. Let E and F be two disjoint sets of variables containing the free variables of s, resp. t.*

If s and t are unifiable, then they have a most general unifier σ, which is E, F-linear. Moreover, if z is any variable in $E \cup F$ that is not free in s or t, then z is not in $\mathrm{dom}\,\sigma$ and not free in $z'\sigma$ for any $z' \in \mathrm{dom}\,\sigma$.

PROOF. Compute σ using the algorithm by Martelli and Montanari (1982). This can be described by the following rewrite rules on finite multisets of equations between terms; we let M be any such multiset, and comma denote multiset union:

(Delete) $M, u \doteq u \to M$

(Decomp) $M, f(u_1, \ldots, u_n) \doteq f(v_1, \ldots, v_n) \to M, u_1 \doteq v_1, \ldots, u_n \doteq v_n$

(Bind) $M, x \doteq v \to M[x := v], x \doteq v$ provided x is not free in v, but is free in M.

We consider that equations $u \doteq v$ are unordered pairs of terms u, v, so that in particular $u \doteq v$ and $v \doteq u$ are the same equation. If s and t are unifiable, then this rewrite process terminates, starting from $s \doteq t$, on a so-called solved form $z_1 \doteq u_1, \ldots, z_k \doteq u_k$; then $\sigma = [z_1 := u_1, \ldots, z_k := u_k]$ is an mgu of $s \doteq t$.

We claim that whenever $M \to M'$, and M is *linear*, in the sense that every variable occurs at most once in M, then M' is linear, too. This is clear for (Delete) and (Decomp), and (Bind) just does not apply. Since the initial multiset $s \doteq t$ is linear, $M_0 = z_1 \doteq u_1, \ldots, z_k \doteq u_k$ is linear, too. Item 1 of Definition 53 is then clear.

Let us say that an equation $u \doteq v$ is *split* if and only if all the free variables of u are in E, and all the free variables of v are in F, or conversely. Let us say that a multiset of equations is split if and only if all its equations are split. We now claim that whenever $M \to M'$ and M is split and linear, then M' is split. This is clear for (Delete) and (Decomp), and (Bind) does not apply on linear multisets.

Let z be any variable. Since M_0 is linear, there is at most one i, $1 \leq i \leq k$, such that z is free in u_i. So there is at most one i such that z is free in $z_i\sigma$. If item 2 were wrong, then there would be two variables x and x', both in E (or, symmetrically, both in F) such that z is free in $x\sigma$ and in $x'\sigma$. Not both x and x' can be in $\{z_1, \ldots, z_k\}$ by the previous remark. Not both x and x' can be outside $\{z_1, \ldots, z_k\}$, otherwise z would be free in $x\sigma = x$ and in $x'\sigma = x'$, entailing that $z = x = x'$. So one of them, say x, is some z_i, $1 \leq i \leq k$, and the other, x', lies outside $\{z_1, \ldots, z_k\}$. Therefore z is free in $z_i\sigma$, and z is free in $x'\sigma = x'$. Since z is free in $z_i\sigma = x\sigma$, z is in F, using the fact that M_0 is split; since z is free in x', $z = x'$ and is therefore in E. This is a contradiction, since no variable is both in E and in F. So item 2 holds.

The final claim is clear, since it is an invariant that all variables occurring in any multiset M obtained from $s \doteq t$ are free in s or t. $\square$

We then study the structure of complete sets of unifiers modulo AC.

Lemma 55 *A* bisimulation *R between two sets A and B is any subset of $A \times B$ such that, for every $x \in A$, there is some $y \in B$ such that $(x, y) \in R$, and for every $y \in B$, there is some $x \in A$ such that $(x, y) \in R$.*

Write $\sum_{k=1}^{p} u_i$ for $u_1 + \ldots + u_p$.

Let $s = \sum_{i=1}^{m} x_i$ and $t = \sum_{j=1}^{n} y_j$ be two linear sums of variables, and assume no x_i equals any y_j. Let E and F be any two disjoint sets of variables containing the *free variables of s, resp. t. Let z_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$ be mn fresh variables, that is, outside $E \cup F$.*

For any bisimulation R between $\{1, \ldots, m\}$ and $\{1, \ldots, n\}$, let σ_R be the substitution such that

$$\begin{aligned} x_i\sigma_R &= \sum_{j/(i,j)\in R} z_{ij} \qquad (1 \leq i \leq m) \\ y_j\sigma_R &= \sum_{i/(i,j)\in R} z_{ij} \qquad (1 \leq j \leq n) \end{aligned}$$

Then the set of all substitutions σ_R, when R ranges over all bisimulations between $\{1, \ldots, m\}$ and $\{1, \ldots, n\}$, is a complete set of unifiers of $s \doteq t$ modulo AC.

Furthermore, for every bisimulation R, σ_R is E, F-linear, and if z is any variable in $E \cup F$ that is not free in s or t, then z is not in $\operatorname{dom} \sigma_R$ and not free in $z'\sigma_R$ for any $z' \in \operatorname{dom} \sigma_R$.

PROOF. First, σ_R is well-defined because, since R is a bisimulation, every right-hand side of the defining equations is a non-empty sum. Second, it is clear that $s\sigma_R = \sum_{i,j/(i,j)\in R} z_{ij} = t\sigma_R$, so σ_R is a unifier of $s \doteq t$ modulo AC.

Conversely, assume σ is any unifier of $s \doteq t$ modulo AC. Write $s\sigma \approx_{\mathrm{AC}} t\sigma$ as a sum $\sum_{k=1}^{N} u_k$, where the terms u_k are not sums. Since this sum equals $\sum_{i=1}^{m} x_i\sigma$ modulo AC, there is a surjective map $f : \{1, \ldots, N\} \to \{1, \ldots, m\}$ such that $x_i\sigma = \sum_{k\in f^{-1}(i)} u_k$ for every i, $1 \leq i \leq m$. Similarly there is a surjective map $g : \{1, \ldots, N\} \to \{1, \ldots, n\}$ such that $y_j\sigma = \sum_{k\in g^{-1}(j)} u_k$, $1 \leq j \leq n$. Let R be the relation defined by $(i, j) \in R$ if and only if $f^{-1}(i) \cap g^{-1}(j) \neq \emptyset$, and let θ be the substitution mapping z_{ij} to $\sum_{k\in f^{-1}(i)\cap g^{-1}(j)} u_k$. Since f and g are surjective, R is a bisimulation. Then $\sigma = \sigma_R\theta$, proving that the set of all σ_R, R a bisimulation, is a complete set of unifiers modulo AC.

To show that σ_R is E, F-linear, note that item 1 is clear. For item 2, let z be any variable. If z is not one of the variables z_{ij}, then z occurs free in $z'\sigma$ if and only if $z' = z$, and z is not one of the variables x_i or y_j. Otherwise, let $z = z_{ij}$, then the only variable x free in s such that z is free in $x\sigma$ is x_i, and the only variable y free in t such that z is free in $y\sigma$ is y_j, proving item 2.

The final claim is clear, in particular from the fact that the variables z_{ij} were chosen outside $E \cup F$. $\square$

Another construction would have been to use the classical Stickel-Fages AC unification algorithm (Stickel, 1981; Fages, 1984), and reason about Diophantine equations with coefficients in $\{0, 1\}$.

Proposition 56 *Any resolvent between two alternation-free Horn clauses of the form* **(4)**, **(5)**, *or* **(6)** *is again alternation-free, and Horn, and a disjunction of variable disjoint clauses of the form* **(4)**, **(5)**, *or* **(6)**.

PROOF. That the resolvent must be Horn is clear. That it is again a disjunction of variable disjoint clauses of the form **(4)**, **(5)**, or **(6)** is by Proposition 1 of Goubault-Larrecq et al. (2004), the stepping stone in proving Proposition 49.

Let us show that it must be alternation-free. To fix notations, let $C \vee +A$ and $-A' \vee C'$ be the premises, and $\sigma \in \mathrm{csu}_{\mathrm{AC}}(A \doteq A')$. The conclusion is $(C \vee C')\sigma$. Let E be the set of free variables of $C \vee +A$, F be that of $-A' \vee C'$. Recall that resolution applies to clauses that have been renamed first so as not to have any free variable in common, so E and F are disjoint. Observe also that the domain $\mathrm{dom}\ \sigma$ of σ contains only variables that are free in s or t.

If $A = A' = q \in \mathcal{Q}$, then σ is the identity. Write C' as $B \Leftarrow B_1, \ldots, B_n$. Then every variable free in B occurs at most once in $B_1, \ldots, B_n$, hence in $C \vee C' = +B \vee -B_1 \vee \ldots \vee -B_n \vee C$, since C and C' share no variable. Moreover, all atoms of $C \vee C'$ are clearly linear, so $C \vee C'$ is alternation-free.

If $A = P(s)$, $A' = P(t)$, where s and t share no free variable, and are linear terms by assumption, then either the only function symbols occurring in s and t are free, so σ is E, F-linear by Lemma 54; or the only function symbol occurring in s and t is $+$, so σ is E, F-linear by Lemma 55. There is no other case, because non-trivial $+$-sums do not unify with terms of the form $f(\ldots)$ with f free.

By Lemma 54 and Lemma 55 again, σ also satisfies: $(*)$ if z is any variable in $E \cup F$ that is not free in s or t, then z is not in $\mathrm{dom}\ \sigma$ and not free in $z'\sigma$ for any $z' \in \mathrm{dom}\ \sigma$.

Since σ is E, F-linear, $x\sigma$ is linear for every variable x free in $C \vee +A$, and either $x\sigma = x$ or the only free variables in $x\sigma$ are variables in F, hence not in E. For every atom of the form $P'(s')$ in $C \vee +A$, it is easy to see that $s'\sigma$ is therefore linear. Similarly, for every atom $P'(t')$ in $-A' \vee C'$, $t'\sigma$ is linear.

Now write C' as $B \Leftarrow B_1, \ldots, B_n$, and let z be some free variable of $B\sigma$. Since z is free in $B\sigma$, z is free in some term $y_0\sigma$, for some free variable y_0 in B, in particular, for some $y_0 \in F$. Since σ is E, F-linear, there is exactly one variable y_0 in F such that z is free in $y_0\sigma$; and y_0 is free in B.

Assume by contradiction that z occurs at least twice in $(-B_1 \vee \ldots \vee -B_n \vee C)\sigma$. If z occurs at all in $C\sigma$, there is a variable x free in C, hence in E, such that z is free in $x\sigma$; since σ is E, F-linear, by item 2, such a variable x is unique if it exists. Similarly, if z occurs at all in $(-B_1 \vee \ldots \vee -B_n)\sigma$, then there is a unique variable

y in F such that z is free in $y\sigma$. We then have three cases:

- Case 1: z occurs twice in $C\sigma$, so x exists, and x occurs twice in C. So x does not occur in $+A = +P(s)$, since C is alternation-free. Since $\mathrm{dom}\ \sigma$ only contains variables that are free in s or t by $(*)$, x cannot be in $\mathrm{dom}\ \sigma$. Since z is free in $x\sigma$, z equals x. Since x is in $E \cup F$ but is not free in s or t, by $(*)$ again, x is not free in $y_0\sigma$ if $y_0 \in \mathrm{dom}\ \sigma$. So either $y_0 \notin \mathrm{dom}\ \sigma$, so $y_0\sigma = y_0$; as $z = x$ is free in $y_0\sigma$, this entails $x = y_0$, contradicting the fact that $x \in E$, $y_0 \in F$ and $E \cap F = \emptyset$. Or $y_0 \in \mathrm{dom}\ \Sigma$, so x is not free in $y_0\sigma$; since $z = x$, z is not free in $y_0\sigma$, a contradiction again.
- Case 2: z occurs twice in $(-B_1 \vee \ldots \vee -B_n)\sigma$, so y exists, and y occurs twice in $-B_1 \vee \ldots \vee -B_n$. Since $C' = B \Leftarrow B_1, \ldots, B_n$ is alternation-free, y is not free in B. Recall that y_0 is in F and z is free in $y_0\sigma$, and that y is the unique variable in F such that z is free in $y\sigma$; so $y = y_0$. But y is not free in B, whereas y_0 is, contradiction.
- Case 3: z occurs once in $C\sigma$ and once in $(-B_1 \vee \ldots \vee -B_n)\sigma$, so x and y both exist, x occurs once in C, and y occurs once in $-B_1 \vee \ldots \vee -B_n$. Since z is free both in $y\sigma$ and in $y_0\sigma$, and $y, y_0 \in F$, by unicity $y = y_0$ (as in Case 2). Since y_0 occurs free in $+B$, y occurs free in $+B$, and also in $-B_1 \vee \ldots \vee -B_n$. If y also occurred free in t, then it would occur twice in $-P(t) \vee -B_1 \vee \ldots \vee -B_n$, contradicting the fact that $-A' \vee C' = +B \vee -P(t) \vee -B_1 \vee \ldots \vee -B_n$ is alternation-free. So y is not free in t.

  We now use an argument similar to Case 1. Since $\mathrm{dom}\ \sigma$ only contains variables that are free in s or t by $(*)$, y cannot be in $\mathrm{dom}\ \sigma$. Since z is free in $y\sigma$, z equals y. Since y is in $E \cup F$ but is not free in s or t, by $(*)$ again, y is not free in $x\sigma$ if $x \in \mathrm{dom}\ \sigma$. So either $x \notin \mathrm{dom}\ \sigma$, so $x\sigma = x$; as $z = y$ is free in $x\sigma$, this entails $y = x$, contradicting the fact that $y \in F$, $x \in E$, and $E \cap F = \emptyset$. Or $x \in \mathrm{dom}\ \sigma$, so y is not free in $x\sigma$; since $z = y$, z is not free in $x\sigma$, a contradiction again.

As all cases lead to a contradiction, z occurs at most once in $(-B_1 \vee \ldots \vee -B_n \vee C)\sigma = (C \vee C')\sigma$. So the resolvent $(C \vee C')\sigma$ is alternation-free. $\square$

We can now refine Proposition 49. Nothing changes except that we have sprinkled the word "alternation-free" throughout, and replaced the set $\mathcal{Q}_0$ of names $\ulcorner B(x) \urcorner$ of ϵ-blocks by the subset $\mathcal{Q}_1$ generated by alternation-free $\mathcal{P}\epsilon$-blocks.

Definition 57 (Candidate, Oracle) *An* alternation-free candidate *is any alternation-free Horn clause of the form $B_1(x_1) \vee \ldots \vee B_n(x_n)[\vee{+q}]$, where the x_is are pairwise distinct, $B_i(x_i)$ is a non-empty block for every i, $1 \leq i \leq n$, and $q \in \mathcal{Q}_1$.*

An alternation-free grey oracle *is any function $\mathcal{O}$ mapping every set of alternation-free clauses of type* **(4)**, **(5)**, *or* **(6)**, *to a set of alternation-free candidates containing all those deducible by grey resolution.*

Proposition 58 *Let $\mathcal{O}$ be any alternation-free grey oracle. Let $\succ_{\mathrm{AC}}$ and* sel *be as in Proposition 49.*

Then white resolution together with the grey oracle rule is complete for every set of alternation-free clauses of type **(4)**, **(5)**, *or* **(6)**. *In other words, for every set S_1 of alternation-free clauses of type* **(4)**, **(5)**, *or* **(6)**, *if S_1 is* AC*-unsatisfiable, then the empty clause can be derived from S_1 by white resolution and the grey oracle rule. Moreover, completeness is retained when removing tautologies, forward subsumed clauses, and ϵ-splitting of clauses not of type* **(6)**.

Notice that we can always restrict alternation-free grey oracles to only output alternation-free clauses. Indeed, we only required that grey oracles output at least all the clauses deducible by grey resolution, but grey resolution only derives alternation-free clauses, starting from alternation-free clauses of type **(4)**, **(5)**, **(6)**, by Proposition 56. The following notion of a.f.-soundness weakens soundness by requiring $\mathcal{O}$ to be sound only on sets of alternation-free clauses.

Definition 59 *An alternation-free grey oracle $\mathcal{O}$ is* a.f.-sound *if and only if, for every set S of alternation-free clauses of type* **(4)**, **(5)**, *and* **(6)**, *$\mathcal{O}(S)$ is a set of alternation-free candidates that are semantic consequences modulo* AC *of the clauses in S.*

Theorem 60 (Main Theorem, Alternation-Free Case) *If there is an a.f.-sound computable alternation-free grey oracle, then* AC*-satisfiability of sets of alternation-free clauses of type* **(4)**, **(5)**, *and* **(6)** *is decidable.*

Now the existence of an a.f.-sound, computable alternation-free grey oracle is equivalent to solving the constant-only case:

Proposition 61 *There is an a.f.-sound computable alternation-free grey oracle if and only if the* AC*-satisfiability of sets of alternation-free* AC^0*-clauses of the form* **(4)**, **(5)**, **(6)** *is decidable.*

PROOF. For any set S of clauses of the above type, let $S_{\mathbf{4},\mathbf{6}}$ be the subset of the clauses of the form **(4)** or **(6)**. To find a grey oracle, we only need it to find at least all consequences by grey resolution of $S_{\mathbf{4},\mathbf{6}}$, not of S. This is because clauses of type **(5)** never resolve with clauses of type **(6)**, and clauses **(4)** and **(6)** only resolve to produce again clauses of type **(4)** or **(6)**, as noticed in Goubault-Larrecq et al. (2004, Proposition 1).

Assume that the constant-only case is decidable. Then define $\mathcal{O}$ as enumerating all alternation-free candidates C (which are finitely many), and returning those such that $S_{\mathbf{4},\mathbf{6}} \models_{\mathrm{AC}} C$. We have already remarked (in justifying that condition 3 of the definition of $\rhd$ was decidable, in Section 7.5) that this could be decided by skolemizing C, and using the decidability of the constant-only case. (Although C is now slightly more general than in Section 7.5, the same argument applies.) Clearly $\mathcal{O}$ is a.f.-sound, computable, and is an alternation-free grey oracle.

Conversely, if there is an a.f.-sound computable alternation-free grey oracle, then by Theorem 60 every set of alternation-free clauses of type **(4)**, **(5)**, and **(6)** is decidable, in particular any subset of AC^0-clauses. $\square$

>From (slight and easy variants of) the latter results, it follows again that intersection-emptiness of AC-standard two-way AC-tree automata is decidable, thus providing another proof of Corollary 45. This is because, by Corollary 27, the constant-only case of AC-standard two-way AC-tree automata is decidable, hence there is a computable grey oracle that is sound for the particular sets of $+$-clauses needed to handle AC-standard two-way AC-tree automata.

More generally, we obtain the following result. This shows that, to settle down the decidability status of intersection-emptiness for two-way AC-tree automata, the only difficulty resides in the constant-only case. As discussed in the conclusion of Verma and Goubault-Larrecq (2005), this may be extraordinarily difficult to deal with, since it includes generalizations of the Petri net reachability problem as (very) particular subproblems.

Proposition 62 *Let $+$ be a fixed associative commutative symbol. Assume that it is decidable whether any given finite set of alternation-free Horn clauses that are either ϵ-blocks, free complex clauses on a fixed signature consisting only of constant symbols, or $+$-clauses, is AC-satisfiable (the so-called* constant-only case*). Then AC-satisfiability of finite sets of alternation-free Horn clauses that are either ϵ-blocks, free complex clauses, or $+$-clauses, is decidable.*

Similar reasoning establishes the following similar theorem.

Proposition 63 *Let $+$ be a fixed associative commutative symbol. Assume that it is decidable whether any given finite set of alternation-free Horn clauses that are either ϵ-blocks, free complex clauses on a fixed signature consisting only of constant symbols, or $+$-pop and $+$-push clauses, is AC-satisfiable. Then intersection-emptiness of two-way AC-tree automata is decidable.*

9 Conclusion

We have classified alternating two-way AC-tree automata according to the decidability of the intersection-emptiness question. Essentially, alternation, general push clauses, and equality constraints between brothers lead to undecidability. On the other hand we were able to give a decision algorithm for two-way AC-tree automata (without alternation), with the restriction that the push clauses on equational symbols must be standard.

The case when conditional $+$-push clauses are included in two-way AC-tree automata is open. Nonetheless we have shown that this reduced to the constant-only case. While this may seem to be a considerable simplification, we have noticed that already intersection-emptiness for the subcase of Petri two-way AC^0-automata included the question of BVASS reachability, which includes that of Petri net reachability. The latter is decidable, but it is not clear at the moment either how to translate Petri two-way AC^0-automata or how to extend the Mayr-Kosaraju algorithm to BVASS, hence to Petri two-way AC^0-automata. However, once this is done, a suitable variant of Proposition 63 should be usable to conclude that a suitable restriction of two-way AC-tree automata (which would naturally be called Petri two-way AC-tree automata) has a decidable intersection-emptiness problem. We conjecture that intersection-emptiness is also decidable for the general case of two-way AC-tree automata, but this is even harder.

Figure 4 sums up our main results.

References

Bachmair, L. and Ganzinger, H. (2001). Resolution theorem proving. In Robinson and Voronkov (2001), chapter 2, pages 19–99.

Bogaert, B. and Tison, S. (1992). Equality and disequality constraints on direct subterms in tree automata. In *Proc. 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS'92)*, pages 161–172. Springer-Verlag LNCS 577.

Boneva, I. and Talbot, J.-M. (2005). Automata and logics for unranked and unordered trees. In *RTA'05*, pages 500–515. Springer-Verlag LNCS 3467.

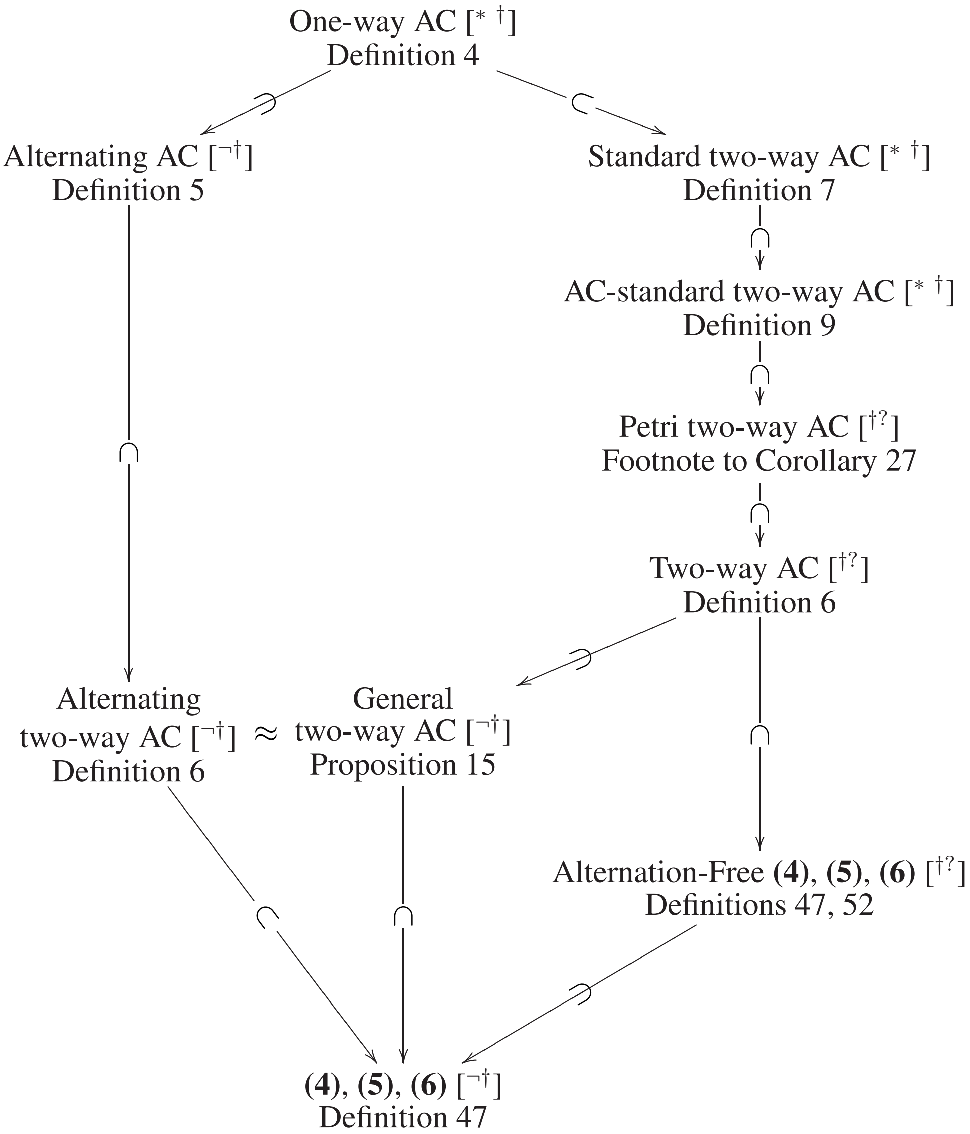
Bouhoula, A. and Jouannaud, J.-P. (1997). Automata-driven automated induction. In LICS-12 (1997), pages 14–25.

Chang, C.-L. and Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Computer Science Classics. Academic Press.

Charatonik, W. and Podelski, A. (1998). Set-based analysis of reactive infinite-state systems. In Steffen, B., editor, *Proc. 1st Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, pages 358–375. Springer-Verlag LNCS 1384.

Comon, H., Cortier, V., and Mitchell, J. (2001). Tree automata with one memory, set constraints and ping-pong protocols. In *Proc. 28th Intl. Colloquium on Automata, Languages, and Programming (ICALP'2001)*, pages 682–693. Springer-Verlag LNCS 2076.

Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. (1997). Tree automata techniques and applications. `www.grappa.univ-lille3.fr/tata/`.



*: closed under union, intersection.
†: intersection-emptiness decidable.
†?: intersection-emptiness decidable if so in the constant-only case.
¬†: intersection-emptiness undecidable.

Fig. 4. Results on the AC-tree automata considered in this paper

Comon, H. and Jacquemard, F. (1997). Ground reducibility is EXPTIME-complete. In LICS-12 (1997), pages 26–34.

Courcelle, B. (1989). On recognizable sets and tree automata. In Nivat, M. and Aït-Kaci, H., editors, *Resolution of Equations in Algebraic Structures*. Academic Press.

Davis, M. D. and Weyuker, E. J. (1985). *Computability, Complexity and Languages*. Academic Press, New York.

de Groote, P., Guillaume, B., and Salvati, S. (2004). Vector addition tree automata.

In *Proc. 19th Annual IEEE Symposium on Logics in Computer Science*. IEEE Computer Society Press. To appear.

Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654.

Dowling, W. F. and Gallier, J. H. (1984). Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 1(3):267–84.

Emerson, E. A. and Jutla, C. S. (1988). The complexity of tree automata and logics of programs (extended abstract). In *Proc. 29th Symposium on Foundations of Computer Science (FOCS'88)*, pages 328–337.

Fages, F. (1984). Associative-commutative unification. In *7th Intl. Conference on Automated Deduction*, pages 194–208. Springer Verlag LNCS 170.

Finkel, A. and Schnoebelen, P. (2001). Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92.

Frühwirth, T., Shapiro, E., Vardi, M. Y., and Yardeni, E. (1991). Logic programs as types for logic programs. In *Proc. 6th Annual IEEE Symposium on Logic in Computer Science (LICS'91)*, pages 300–309. IEEE Computer Society Press.

Gécseg, F. and Steinby, M. (1997). Tree languages. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages*, volume 3, pages 1–68. Springer Verlag.

Genet, T. (1998). Decidable approximations of sets of descendants and sets of normal forms. In Nipkow, T., editor, *Proc. of the 9th Intl. Conference on Rewriting Techniques and Applications (RTA'98)*, pages 151–165. Springer Verlag LNCS 1379.

Genet, T. and Klay, F. (2000). Rewriting for cryptographic protocol verification. In *17th Intl. Conference on Automated Deduction (CADE-17)*, pages 271–290. Springer Verlag LNCS 1831.

Ginsburg, S. and Spanier, E. H. (1966). Semigroups, Presburger formulas and languages. *Pacific Journal of Mathematics*, 16(2):285–296.

Goré, R., Leitsch, A., and Nipkow, T., editors (2001). *1st Intl. Joint Conference on Automated Reasoning (IJCAR'01)*, Siena, Italy. Springer Verlag LNAI 2083.

Goubault-Larrecq, J. (2000). A method for automatic cryptographic protocol verification. In *Formal Methods in Parallel Programming Theory and Applications (FMPPTA'2000), 15th IPDPS Workshops*, pages 977–984. Springer-Verlag LNCS 1800.

Goubault-Larrecq, J. (2002). Higher-order positive set constraints. In Bradfield, J., editor, *15th Annual Conf. of the European Association for Computer Science Logic (CSL'02)*, pages 473–489. Springer Verlag LNCS 2471.

Goubault-Larrecq, J. (2003). Résolution ordonnée avec sélection et classes décidables de la logique du premier ordre. Lecture notes for the course "démonstration automatique et vérification de protocoles cryptographiques" (with Hubert Comon-Lundh), DEA "programmation". 70 pages, `http://www.lsv.ens-cachan.fr/~goubault/SOresol.ps`. In French.

Goubault-Larrecq, J., Roger, M., and Verma, K. N. (2004). Abstraction and resolution modulo AC: How to verify Diffie-Hellman-like protocols automatically.

Journal of Logic and Algebraic Programming. To appear. Available as LSV Research Report LSV-04-7, Mar. 2004, `http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/rr-lsv-2004-7.rr.ps`.

Goubault-Larrecq, J. and Verma, K. N. (2002). Alternating two-way AC-tree automata. Research Report LSV-02-11, LSV, ENS de Cachan. Available at `http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/rr-lsv-2002-11.rr.ps`.

Hopcroft, J. and Pansiot, J. J. (1979). On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8:135–159.

Ibarra, O. H., Su, J., Dang, Z., Bultan, T., and Kemmerer, R. A. (2001). Counter machines and verification problems. *Theoretical Computer Science*. To appear.

Jouannaud, J.-P. (1995). Rewrite proofs and computations. In Schwichtenberg, H., editor, *Proof and Computation*, volume 139 of *NATO series F: Computer and Systems Sciences*, pages 173–218. Springer Verlag.

Kaji, Y., Fujiwara, T., and Kasami, T. (1997). Solving a unification problem under constrained substitutions using tree automata. *Journal of Symbolic Computation*, 23(1):79–117.

Karp, R. M. and Miller, R. E. (1969). Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195.

Kosaraju, S. R. (1982). Decidability of reachability in vector addition systems. In *Proc. 14th Annual ACM Symposium on the Theory of Computing (STOC'82)*, pages 267–281.

Lambert, J.-L. (1992). A structure to decide reachability in Petri nets. *Theoretical Computer Science*, 99(1):79–104.

Li, P. (1988). Pattern matching in trees. Master's Thesis CS-88-23, University of Waterloo.

LICS-12 (1997). *Proc. 12th Annual IEEE Symposium on Logic in Computer Science (LICS'97)*. IEEE Computer Society Press.

Lipton, R. (1976). The reachability problem requires exponential space. Technical Report 62, Dept. Computer Science, Yale University.

Lugiez, D. (1998). A good class of tree automata. Application to inductive theorem proving. In *Proc. 25th Intl. Colloquium on Automata, Languages, and Programming (ICALP'98)*, pages 409–420. Springer-Verlag LNCS 1443.

Lugiez, D. (2003). Counting and equality constraints for multitree automata. In *Proc. 6th Conference on Foundations of Software Science and Computation Structures (FoSSaCS'03), European Joint Conferences on Theory and Practice of Software (ETAPS'03)*, Springer-Verlag LNCS 2620, pages 328–342.

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

Martelli, A. and Montanari, U. (1982). An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282.

Mayr, E. W. (1984). An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13:441–460.

Mayr, R. and Rusinowitch, M. (1998). Reachability is decidable for ground AC rewrite systems. In *Proceedings of the 3rd Intl. Workshop on Verification of*

Infinite State Systems (INFINITY'98), pages 53–64, Aalborg, Denmark. Technical Report TUM-I9825, Technische Universität München. `http://www.informatik.uni-freiburg.de/~mayrri/ac.ps`.

Minsky, M. L. (1961). Recursive unsolvability of Post's problem of "tag" and other topics in the theory of Turing machines. *Annals of Mathematics, Second Series*, 74(3):437–455.

Monniaux, D. (1999). Abstracting cryptographic protocols with tree automata. In *Proc. 6th Intl. Static Analysis Symposium (SAS'99)*, pages 149–163. Springer-Verlag LNCS 1694.

Niehren, J. and Podelski, A. (1993). Feature automata and recognizable sets of feature trees. In *Proc. 4th Intl. Conference on Theory and Practice of Software Development (TAPSOFT'93)*, pages 356–375. Springer-Verlag LNCS 668.

Ohsaki, H. (2001). Beyond regularity: Equational tree automata for associative and commutative theories. In *14th Annual Conf. of the European Association for Computer Science Logic (CSL'01)*, pages 539–553. Springer-Verlag LNCS 2142.

Ohsaki, H. and Takai, T. (2002). Decidability and closure properties of equational tree languages. In Tison, S., editor, *Proceedings of the 13th Conference on Rewriting Techniques and Applications (RTA'02)*, pages 114–128. Springer-Verlag LNCS 2378.

Okhotin, A. (2001). Conjunctive grammars. *Journal of Automata, Languages and Combinatorics*, 6(4):519–535.

Parikh, R. J. (1966). On context-free languages. *Journal of the Association for Computing Machinery*, 13(4):570–581.

Peltier, N. (1997). Tree automata and automated model building. *Fundamenta Informaticae*, 30(1):59–81.

Plotkin, G. (1972). Building in equational theories. *Machine Intelligence*, 7:73–90.

Reutenauer, C. (1993). *Aspects Mathématiques des Réseaux de Petri*. Masson.

Riazanov, A. and Voronkov, A. (2001). Vampire 1.1 (system description). In Goré et al. (2001), pages 376–380.

Robinson, J. A. and Voronkov, A., editors (2001). *Handbook of Automated Reasoning*. North-Holland.

Roger, M. (2003). *Raffinements de la résolution et vérification de protocoles cryptographiques*. PhD thesis, ENS de Cachan. In French.

Sacerdote, G. S. and Tenney, R. L. (1977). The decidability of the reachability problem for vector addition systems. In *Proc. 9th Annual ACM Symposium on the Theory of Computing (STOC'77)*, pages 61–76.

Schimpf, K. M. and Gallier, J. H. (1985). Tree pushdown automata. *Journal of Computer and System Sciences*, 30(1):25–40.

Seidl, H. (1994). Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52(2):57–60.

Seidl, H., Schwentick, T., and Muscholl, A. (2003). Numerical document queries. In *22nd ACM Symposium on Principles of Database Systems (PODS'03)*, pages 155–166, San Diego, CA, USA.

Shepherdson, J. C. (1959). The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3:199–201.

Slutzki, G. (1985). Alternating tree automata. *Theoretical Computer Science*, 41:305–318.

Stickel, M. (1981). A unification algorithm for associative-commutative functions. *Journal of the Association for Computing Machinery*, 28(3):423–434.

Thomas, W. (1990). Automata on infinite objects. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier Science.

Verma, K. N. (2003a). *Automates d'arbres bidirectionnels modulo théories équationnelles*. PhD thesis, ENS de Cachan. In English, with French abstracts.

Verma, K. N. (2003b). On closure under complementation of equational tree automata for theories extending AC. In *Proc. 10th Intl. Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'2003)*, pages 183–195. Springer-Verlag LNAI 2850.

Verma, K. N. (2003c). Two-way equational tree automata for AC-like theories: Decidability and closure properties. In *Proceedings of the 14th Conference on Rewriting Techniques and Applications (RTA'2003)*, pages 180–196. Springer-Verlag LNCS 2706.

Verma, K. N. (2004). Alternation in equational tree automata modulo XOR. In *24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, volume 3328 of *LNCS*, Chennai, India. Springer-Verlag.

Verma, K. N. and Goubault-Larrecq, J. (2005). Karp-Miller trees for a branching extension of VASS. *Discrete Mathematics and Theoretical Computer Science*, 7(1):217–230.

Verma, K. N., Seidl, H., and Schwentick, T. (2005). On the complexity of equational Horn clauses. In Nieuwenhuis, R., editor, *20th International Conference on Automated Deduction (CADE'05)*, volume 3632 of *LNCS*, pages 337–352, Tallinn, Estonia. Springer-Verlag.

Voronkov, A. (2001). Algorithms, datastructures, and other issues in efficient automated deduction. In Goré et al. (2001), pages 13–28.

Weidenbach, C. (2001). Combining superposition, sorts and splitting. In Robinson and Voronkov (2001), chapter 27, pages 1965–2013.

A Deciding Emptiness of Standard Two-Way AC^0-Automata

The following proposition states in particular that emptiness is decidable for standard two-way AC^0-automata. This is a particular case of Corollary 27, which implies in particular that intersection-emptiness of standard two-way AC^0-automata is decidable. We include this proposition, as its proof is simpler, the decision algorithm is straightforward, and we obtain an explicit complexity bound.

Introduce the following notation: $\langle n \rangle$, for any $n \geq 1$, denotes the sum of n distinct variables. The idea is that the only ground instances of $\langle n \rangle$ are sums of at least n

constants a_i (not necessarily distinct).

Proposition 64 *The AC-satisfiability of sets of free pop clauses, of clauses (14)–(17), and generalized query clauses*

$$\bot \Leftarrow P(\langle n \rangle)$$

is decidable and in NP.

In particular, it is decidable in NP whether, given any standard two-way AC^0-automaton $\mathcal{A}$ and a state P, the language $L_P(\mathcal{A})$ is empty.

PROOF. It suffices to show that input resolution, with eager linear subsumption and splitting, terminates.

Input resolution only generates new negative clauses. We claim that, starting from S_0, the only negative clauses generated by input resolution with eager splitting are generalized query clauses.

- Resolving $\bot \Leftarrow P(\langle n \rangle)$ with a free pop clause $P(f(x_1, \ldots, x_m)) \Leftarrow P_1(x_1), \ldots, P_m(x_m)$ yields $\bot \Leftarrow P_1(x_1), \ldots, P_m(x_m)$ (provided $n = 1$), which splits as the m clauses $\bot \Leftarrow P_i(\langle 1 \rangle)$, $1 \leq i \leq m$.
- Resolving $\bot \Leftarrow P(\langle n \rangle)$ with a $+$-pop clause $P(x + y) \Leftarrow P_1(x), P_2(y)$. The intuition is that unifying $x + y$ with $\langle n \rangle$ means splitting $\langle n \rangle$ (a sum of at least n constants) in two sums; any unifier must then map x to $\langle n_1 \rangle$ and y to $\langle n_2 \rangle$ with $n_1 + n_2 = n$; resolution generates $\bot \Leftarrow P(\langle n_1 \rangle), P(\langle n_2 \rangle)$, which splits in $\bot \Leftarrow P(\langle n_1 \rangle)$ and $\bot \Leftarrow P(\langle n_2 \rangle)$.

  Formally, use Lemma 55; writing $\langle n \rangle$ as $x_1 + \ldots + x_n$, we get that the elements of a complete set of AC-unifiers of $\langle n \rangle$ and $x + y$ are the substitutions σ_R, where R ranges over all bisimulations between $\{1, \ldots, n\}$ and $\{1, 2\}$. Observe that σ_R maps x to $\sum_{i/(i,1) \in R} z_{ij}$ and y to $\sum_{i/(i,2) \in R} z_{ij}$, where the variables z_{ij} are fresh, and $\sum_{i,j} z_{ij} \approx_{\mathrm{AC}} \langle n \rangle$. Let n_j be the number of indices i such that $(i, j) \in R$, $j \in \{1, 2\}$. In particular $n_1 + n_2 = n$.

  The resulting resolvent is $\bot \Leftarrow P_1(\sum_{i/(i,1) \in R} z_{ij}), P_2(\sum_{i/(i,2) \in R} z_{ij})$. Since no z_{ij} free in the first atom is free in the second one, this splits. The resulting clauses are then $\bot \Leftarrow P_1(\langle n_1 \rangle)$ and $\bot \Leftarrow P_2(\langle n_2 \rangle)$.
- Resolving $\bot \Leftarrow P(\langle n \rangle)$ with a base clause $P(a_i)$ only succeeds when $n = 1$, and the resulting resolvent is the empty clause.
- Resolving $\bot \Leftarrow P(\langle n \rangle)$ with an ϵ-clause $P(x) \Leftarrow P_1(x)$ yields $\bot \Leftarrow P_1(\langle n \rangle)$.
- Resolving $\bot \Leftarrow P(\langle n \rangle)$ with a standard $+$-push clause $P(x) \Leftarrow P_1(x+y)$ yields $\bot \Leftarrow P_1(\langle n + 1 \rangle)$.

If any branch of the tableau obtained by input resolution, linear subsumption, and splitting were infinite, then there would be infinitely many generalized query clauses on this branch. By the pigeonhole principle, there would be a predicate symbol P and an infinite sequence of integers $n_1 < n_2 < \ldots$ such that all generalized query clauses $\perp \Leftarrow P(n_i)$ are on the branch. But $\perp \Leftarrow P(n_1)$ subsumes them all linearly, so only finitely of them can have been generated and survived linear subsumption.

Let us evaluate the complexity of the process. Note that the size of $P(\langle n \rangle)$ is linear in n (i.e., n is coded in unary, as a sum of n distinct variables). Note that, by using linear subsumption, for every predicate symbol P, there is at most one clause $\perp \Leftarrow P(\langle n \rangle)$ in any branch S at any time. Let $N(S)$ be the largest integer n such that $\perp \Leftarrow P(\langle n \rangle)$ is in S for some predicate P (0 if none), let $\sum(S)$ be their sum, $k(S)$ be the number of predicates P in S_0 such that no clause $\perp \Leftarrow P(\langle n \rangle)$ is present in S, and K the number of predicates in S_0. Define the measure $\mu(S)$ as $k(S)N(S)+(k(S)+1)(k(S)+2)/2+\sum(S)$, and note that $\mu(S)$ is polynomial in the size of S. We can now check that $\mu(S)$ always decreases from any branch to any of its descendants; the idea is that $\sum(S)$ decreases strictly when a new resolvent $\perp \Leftarrow P(\langle n \rangle)$ is added and there was already some clause $\perp \Leftarrow P(\langle n' \rangle)$, because the new clause can be added only if $n < n'$ and hence the older clause gets subsumed; while generating a new resolvent $\perp \Leftarrow P(\langle n \rangle)$ when no clause $\perp \Leftarrow P(\langle n' \rangle)$ is present (then necessarily $n \leq N(S) + 1$, the worst case being when we resolve with a standard $+$-push clause) decreases $k(S)$ by one, increases $\sum(S)$ by at most $N(S) + 1$, and increases $N(S)$ by at most one. In other words, if S' is any new branch obtained this way from S, $k(S') = k(S) - 1$, $\sum(S') \leq \sum(S) + N(S) + 1$, and $N(S') \leq N(S) + 1$. So

$$
\begin{aligned}
\mu(S') &= k(S')N(S') + \frac{(k(S')+1)(k(S')+2)}{2} + \sum(S') \\
&\leq (k(S)-1)(N(S)+1) + \frac{k(S)(k(S)+1)}{2} + \sum(S) + N(S) + 1 \\
&= k(S)N(S) - N(S) - 1 + k(S) + \frac{k(S)(k(S)+1)}{2} + \sum(S) + N(S) + 1 \\
&= k(S)N(S) - N(S) - 2 + \frac{(k(S)+1)(k(S)+2)}{2} + \sum(S) + N(S) + 1 \\
&= \mu(S) - 1 < \mu(S)
\end{aligned}
$$

Since generating resolvents until a new one is found takes time polynomial in the size of S, which is bounded by the number of non-negative clauses in S_0 plus $\sum(S)$, and since we can only decrease $\mu(S)$ polynomially many times, every branch is eventually saturated in polynomially many steps. Since AC-satisfiability of S is equivalent to the existence of a saturated branch not containing the empty clause in the tableau, the problem is in NP. $\square$

We do not know whether the problem is in P, or NP-complete, or inbetween.