Extended Tree Automata Models for the Verification of Infinite State Systems

Mémoire d'Habilitation à Diriger des Recherches

Florent Jacquemard

Members of the jury:

- Jean-Pierre Jouannaud (reviewer)
- Christoph Löding
- Denis Lugiez
- Luc Segoufin
- Helmut Seidl (reviewer)
- Sophie Tison (reviewer)

November 2011

Contents

Introduction

This document presents some research work which has mostly been done since mid-2007, when I joined the research team DAHU at LSV, on Databases and Verification. The main goal of this research is the study of several models of state machines, all of them extending the same formalism: the classical tree automata, as defined in the first chapter of [Comon et al., 2007], and their application in various reasoning tasks, such as static analysis of programs or systems, typing, verification of the consistency of specifications, model checking...

Trees are a natural data structure, widespread in computer science, for instance for the representation of hierarchical or nested data structures, *e.g.* filesystems, for specific algorithms (binary search trees, distributed algorithms), for an abstract model for semi-structured data, sometimes also called Web data, for an algebraic presentation of recursive processes, as terms in logic... When it comes to reasoning on systems manipulating trees, or modeled by trees, it is crucial to have finite representation of infinite sets of trees. Tree automata are finite state machines providing such a representation, acknowledged as suitable for a use in reasoning tasks: they are a well established theoretical model, in close relationship with logic, they enjoy good compositional properties and efficient decision algorithms. In particular, tree automata are used at the core of systems for software and hardware verification (*e.g.* Mona [Klarlund and Møller, 2001]) and theorem proving (*e.g.* SPASS [Weidenbach et al., 2009]).

However, tree automata have also some severe limitations in expressiveness, which we describe below. Some extensions have been proposed in order to improve the model while trying to preserve good properties. We present in this document several such extensions, their properties and the advents of their use in symbolic verification of systems and programs.

Tree automata are good at reasoning tasks...

In the context of formal verification, tree structures are appropriate for the symbolic representation of the configurations of several kinds of systems. Consider for instance the following toy example of a recursive *functional program* defining an append function on lists (see also Section II.1)

$$\begin{aligned} \mathsf{app}(\mathsf{nil}, y) &= y, \\ \mathsf{app}\big(\mathsf{cons}(x, y), z\big) &= \mathsf{cons}\big(x, \mathsf{app}(y, z)\big) \end{aligned}$$

In order to check that this function has the expected behavior on every pair of lists ℓ_1, ℓ_2 built with nil (empty list), the constructor symbols cons, and possibly some other symbols for building the elements of the lists, we must consider an infinite set L_{in} of terms of the form $\mathsf{app}(\ell_1, \ell_2)$, which can be characterized by a tree automaton (such a tree language is called *regular*). In this approach, sometimes called *regular model checking* [Bouajjani et al., 2000], both the values (lists) and the intermediate steps of computation are presented as terms. The closure of L_{in} by the two above equations, applied from left to right, it also a regular tree language (see *e.g.* [Genet and Tong, 2001]). This latter language, let us call it L^*, can be analyzed in order to verify that the above definition of the function app is correct. For instance, one can check that the intersection of L^* with a regular tree language L_{err}, representing erroneous configurations, is empty. This is possible because the class of regular tree languages is effectively closed under intersection (with a polynomial construction) and the emptiness of the language of a given tree automaton is decidable (in linear time).

A similar algebraic approach can be followed for the analysis of *imperative processes* with procedure call and creation of concurrent threads, see *e.g.* [Seidl, 2009; Bouajjani et al., 2006]. In process calculi modeling concurrency, processes are represented by terms over a signature containing operators for the composition of processes. For instance, the terms $s \cdot t$ and $s \| t$ represent processes obtained respectively by the sequential and parallel composition of the processes s and t. The evolution of the processes can be defined by rewrite rules between terms. For instance, the following rule describes the creation of a new thread c at a program point a: $\mathsf{a} \to \mathsf{a} \| \mathsf{c}$. By iterated application of this rule, we can obtain an infinite set of reachable terms of the form $\mathsf{a} \| \mathsf{c} \| \ldots \| \mathsf{c}$, which is a regular tree language. In Section II.1 we present also an example of verification of networks of concurrent processes with a tree topology and without bound on the tree size [Abdulla et al., 2002].

XML documents are commonly represented as labelled trees, and the typing formalism in use for XML are all subcases of automata on unranked trees [Murata et al., 2005]. For instance the following set of rules D_{hos} defines a regular set of trees, each of them containing the base of patients of an hospital. These rules associate to every symbol in an alphabet Σ a regular expression over Σ describing the valid sequences of its children, or a content *str*, denoting an arbitrary string. These are similar to declarations of a *Document Type Definition* (DTD, see also page 63), a standard XML schema language.

$$
\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
\mathsf{hospital} & \to & \mathsf{patient}^* & \mathsf{name} & \to & \mathit{str} & \mathsf{ssn} & \to & \mathit{str} \\
\mathsf{patient} & \to & \mathsf{name}, \mathsf{ssn}, \mathsf{treatment} & \mathsf{mref} & \to & \mathit{str} & \mathsf{diagnosis} & \to & \mathit{str} \\
\mathsf{treatment} & \to & \mathsf{mref}, \mathsf{diagnosis}, \mathsf{date} & \mathsf{date} & \to & \mathit{str} & & &
\end{array}
$$

The problem of *static typechecking* programs defining tree transformations consist in verifying that a program always converts valid source documents into valid output trees, where validity is expressed with respect to types defined by tree automata. This problem reduces in some cases to decision problems for tree automata [Milo et al., 2003]. In Section II.2.4, we present for instance a typechecking procedure for the iter-

ations of XML update primitives defined in the W3C recommendation [Robie et al., 2011].

To summarize, standard tree automata are a well-suited formalism for performing formal verifications of infinite state systems, thanks to the following relevant properties

- they have a good expressiveness, and are in particular closely related to logics, like second-order monadic of the tree, in which properties of systems can be expressed,
- they enjoy good compositional properties, in particular, regular languages are closed under all Boolean operations,
- efficient decision algorithms exist for the main problems, *e.g.* emptiness of the language of a given tree automaton is decidable in linear time, and membership (whether a given tree is accepted by a given tree automaton) is decidable in quadratic time.

...but tree automata are sometimes limited.

Tree automata also have some limitations which restrict their use for verification and make necessary some approximations or extensions.

Let us come back to the previous example on the analysis of functions on lists, and in particular to the definition of the language L_{err} of erroneous configurations. It can be defined by a finite set of forbidden patterns. For instance, the set of terms containing the pattern $\mathsf{app}(\mathsf{app}(y_1, y_2), y_3)$ is definable by a tree automaton, because the pattern is *linear* (it does not contain a variable with multiple occurrences). But this is not the case for non linear patterns such as $\mathsf{cons}(x, \mathsf{cons}(x, y))$ which characterizes stuttering lists (this pattern could be interesting for the verification of a more sophisticated definition of app than the above one). For the detection of non-linear patterns, some extensions of standard tree automata are necessary. What is needed for this purpose is the ability to perform test of equalities between the subterms at multiple positions of the same variable in a pattern. Note that the positions tested are at a bounded distance, hence the tests can be performed *locally*, during the computation steps of an automaton.

The algebraic presentation of imperative programs as terms usually requires taking into account equations between terms which do not preserve regular tree languages. This is the case for instance of associativity (A) for the sequential composition operator $\cdot$ or the combination of associativity and commutativity (C), for the parallel composition operator $\|$. The closure of a regular tree language modulo (A) or (AC) (the combination of (A) and (C)) is in general not regular. Hence we need an extension of tree automata model able to characterize languages modulo equational theories such as above, in order to reason on imperative programs with algebraic properties, see *e.g.* [Ohsaki, 2001; Ohsaki et al., 2005; Dal Zilio and Lugiez, 2006].

The problem is that handling equations may complicate emptiness decision procedures. Alternatively, we shall also study the closure of regular tree languages under term rewriting systems (such as the above language L^*). In some cases where this closure is not regular, it is possible to extend the standard tree automaton model while retaining good decision properties.

XML documents are often given along with an *XML schema* which can be used to ease querying and restructuring. One part of a schema is a typing mechanism (*e.g.* a DTD such as the previous D_{hos}), which is always definable by tree automata, while a second part of a schema is made of integrity constraints restricting the structure of documents. A classical example, common in relational databases [Abiteboul et al., 1995] (from which many XML documents are generated), is the *key constraints*, which express that some positions in the documents are uniquely identified by the value of some attributes. For instance, one might want to impose, in the definition D_{hos} of the type hospital, that the string below two distinct positions labeled by ssn are different (*i.e.* the social security number is an identifier). In a DTD, this can be expressed by declaring the string below ssn as an ID attribute. This unary unary key constraint can also be written $\mathsf{ssn}[\downarrow] \rightarrow \mathsf{ssn}$ following the notations of [Buneman et al., 2001] presented in Section I.3.6 (every two positions labeled by ssn must be the same if the two strings immediately below them are equal). Similarly, a *denial constraint* $\mathsf{ssn}[\downarrow] \neq \mathsf{mref}[\downarrow]$ expresses that two strings below respectively a ssn and a mref position are different. We have already mentioned that standard tree automata capture the type mechanisms of all XML schemes in use, but they are inadequate to express integrity constraints as above. In order to fill this gap, we need an external mechanism for expressing constraints of equality and disequality between subterms at positions specified for instance by labels. Note that, unlike the case of pattern matching, the positions to be tested can be at unbounded distance. Therefore, the integrity constraints cannot be verified using local test. Instead, we need a *global* test, performed at the end of a computation.

In conclusion, in many situations, we need to find extensions of standard tree automata that should preserve as much as possible their good properties. They should be as natural as possible, and preferably have a correspondence with logic. They should enjoy closure properties, at least closure by intersection with regular tree languages, and they should also have decidable membership and /or emptiness problems. A tree automaton model enjoying all these desirable properties is unknown, and unlikely to exists in our opinion. Therefore, we shall consider in this document several extensions of the tree automaton model fitting with different kinds of applications.

Plan

This document is organized in three parts. The two first parts present studies of two kinds of properties of various tree automata models: static properties in the first part (Boolean closure and decision problems), and dynamic properties (closure under tree transformations) in the second part. The third part concludes the document with a presentation of two research proposals.

In Part I, we present various classes of extended tree automata and some studies of the properties previously mentioned (composition, emptiness or membership decision...) and we describe applications to first order theorem proving and constraint solving.

The first automata model, in Section I.1, is a model of tree automata extended with local constraints, testing equalities and disequalities between subtrees at a bounded distance in the input tree. This enables in particular non-linear pattern matching. The most general class is undecidable, and we present several decidable subclasses. Some applications of these models to inductive first-order theorem proving are presented in Section I.1.5.

In Section I.2, we introduce tree automata with equational constraints modulo equational theories. These models combine local equality constraints and closure under equational theories. This represents a double challenge because the presence of constraints complicates both the results of emptiness decision and of closure. It is solved with a uniform presentation based on first order Horn clauses and decision procedures based on classical theorem proving techniques. We also present in Section I.2.5 tree automata extended with an auxiliary memory and constraints for comparing the memory contents. These automata are applied to the verification of communicating concurrent processes in Section I.2.7.

The third family of models presented in Part I (Section I.3) is tree automata extended with global constraints which can express XML integrity constraints, such as keys. We present decision results for several subclasses and a correspondence with an extension of the monadic second-order logic of the tree (Section I.3.6).

Part II is devoted to the problem of regular model checking, i.e. the computation of the closure of a tree language by iteration of tree transformation rules, in particular term rewriting rules, and its application to the verification of infinite state systems where (infinite) sets of configurations are represented by tree languages and the dynamics are represented by some transition systems.

Section II.1 is concerned with standard term rewriting systems (TRS). Many works treat the problem of the closure of regular tree languages by TRS. We consider the much less well studied problem of the closure of extended tree automata languages, including the case of tree automata with constraints. We show also how tree automata techniques helped to solve open decision problems in term rewriting theory.

We present in Section II.2 different classes of automata for unranked trees. They define languages of trees strongly related to the associative and associative commutative closures of regular sets of terms (see Sections II.2.1 and II.2.6). We study the closure of these languages by unranked tree rewrite systems (a formalism much less studied than TRS), and show how the results obtained can be applied to the verification of XML updates (Section II.2.4) and XML read/write access control policies (Section II.2.5).

In Section II.3, we consider the problem of the closure of tree automata languages under rewriting with various strategies. These strategies enable more precise representation of transitions of systems, for instance XML transformations where the

positions of transformation are selected with some external mechanism. Using strategies, one can obtain better (II.3.1) or worth (II.3.2) results than with plain rewriting *wrt* closure of tree languages, according to the cases.

Several open problems and perspectives are presented along the document. In the final Part III, we propose, as a conclusion, two longer research subjects. The first proposal III.1 is the study of extensions of tree automata with other symbolic constraints than equalities or disequalities between subtrees. The second one III.2 is a comparison between tree automata with constraints and various automata and logics for *data trees*, which are trees labeled over an infinite alphabet used to represent XML documents carrying data from an infinite domain.

What this document is not about

Before giving more details on the contents of the document, let us precise first what is not in this document. First, all the automata presented in this document compute on finite labelled trees. The automata on infinite trees, which are used widely *e.g.* in the context of verification and game theory, are out of the scope of this memoir.

Moreover, we consider tree automata computing with parallel moves of several heads in an input tree: bottom-up from children to parent or top-down from parent to children. Automata computing sequentially in trees, like tree-walking automata are not considered in this document, neither are automata computing on on the textual form of XML documents (like the visibly pushdown automata of [Alur and Madhusudan, 2004]). See [Schwentick, 2007] for a survey of these topics.

Finally, we purposely focused on formalisms defining languages more expressive than regular tree languages (*i.e.* the languages of standard tree automata). There is a whole thread of research about the automata languages below the regular tree sets, like for instance the languages definable in first-order logic, see *e.g.* [Place, 2010] for recent developments.

Outline of Part I: Classes of Extended Tree Automata

The first extension of tree automata presented in this document is defined by adding *local constraints* to the transition rules in order to test, at each computation step, some equalities and disequalities between "close" subtrees of the input tree (Section I.1). These models were introduced as decision tools in the context of automated theorem proving, because of their strong connexion to term rewriting systems [Dershowitz and Jouannaud, 1990] (a rule-based formalism for describing calculations on terms of first order logic, by pattern matching and replacement).

For instance, in a somewhat old work [Comon and Jacquemard, 1997, 2003] (Section I.1.3), we studied with Hubert Comon the complexity of the emptiness problem for a tree automata model with local disequality constraints that are able to represent the set of *normal forms* of term rewriting systems (the terms that cannot be evaluated anymore). More recently, we have refuted with Michael Rusinowitch and Laurent Vigneron [Jacquemard et al., 2008a] (see Section I.2.4) an old conjecture

on the emptiness decision for a larger class of such automata, called *reduction automata* [Dauchet et al., 1995]. Some results regarding the problem of the decision of *regularity* (is the language of a given extended automaton also a language of a standard tree automaton?), derived from a long version of [Barguñó et al., 2010] currently under submission, are also presented in the document.

With Adel Bouhoula [Bouhoula and Jacquemard, 2008, 2006, 2007, 2011] (Section I.1.5) we have developed a framework for automating proofs by induction, generalizing an idea of [Bouhoula and Jouannaud, 2001], where tree automata with local constraints, characterizing the languages of normal forms, are used both as induction schemes and as decision tools.

It is known that tree automata can be presented declaratively, as logic programs (more precisely as a set of first order Horn clauses) [Frühwirth et al., 1991; Nielson et al., 2002]. This presentation, followed in Section I.2, allows one to apply well established automated deduction techniques and tools for the decision problem of tree automata. With Michael Rusinowitch and Laurent Vigneron [Jacquemard et al., 2006], we have proposed a clausal definition of different models of tree automata with local equality constraints and computing modulo equational theories (Sections I.2.1-I.2.4). We use a *paramodulation* calculus with basic and ordered strategies and with selection [Bachmair et al., 1995; Nieuwenhuis and Rubio, 2001] in order to ensure the termination of decision algorithms, which have been implemented [Jacquemard et al., 2008b]. In a much earlier work with Christoph Meyer and Christoph Weidenbach [Jacquemard et al., 1998], we had applied a similar approach to tree automata with simpler local constraints and modulo *flat* equational theories (the terms in the equations of the theory are of height at most one), using a different *superposition* calculus. This method has been implemented in the system SPASS [Weidenbach et al., 2009].

In addition, following an internship of Nicolas Perrin, we proposed with him and Hubert Comon [Comon-Lundh et al., 2007] a tree automata model called *visibly with one memory* (presented in Section I.2.5 as sets of Horn clauses), which are extended by an auxiliary memory containing a tree and whose expressiveness lies between the class of regular tree languages and the class of context-free tree languages. Unlike the latter, the class of languages defined by visibly tree automata with memory is closed under intersection and complement. These automata have been extended with constraints allowing, in the calculations, some tests on the contents of memory and local tests on the input tree [Comon-Lundh et al., 2007; Comon-Lundh et al., 2008].

These clausal models of constrained tree automata can be used for checking safety properties of concurrent processes exchanging asynchronously some composite messages (which are labeled trees of unbounded size) via unreliable communication channels, in a model related to the applied π-calculus [Abadi and Fournet, 2001], see Section I.2.7 and [Jacquemard et al., 2008a].

The extensions of tree automata with *global constraints* are presented in Section I.3. These constraints are tested only once, at the end of a computation of the automaton on a tree, and consist in a combination of tests of equalities and disequalities between subtrees whose respective positions are defined by the automaton's computation. A fundamental difference with the local constraints previously

mentioned is that the distance between the positions tested is arbitrary.

The first tree automata model of this type, called TAGED (see Section I.3.1), was introduced in [Filiot et al., 2007] in connection with the decision of the satisfiability for the spatial logic TQL, which permits ones to express queries on XML documents. With Francis Klay and Camille Vacher [Jacquemard et al., 2009] we have studied independently a subclass of TAGED called *rigid tree automata* (Section I.3.2) which has good properties of decision and expressiveness. An application to the verification of communicating processes is presented in [Jacquemard et al., 2011a]. A combination of the models of [Jacquemard et al., 2008a] and [Jacquemard et al., 2011a] would be interesting in this context, see *e.g.* [Affeldt and Comon-Lundh, 2009].

The problem of decidability of emptiness for TAGED remained open until 2010. We have solved it with Luis Bargunó, Carlos Creus, Guillem Godoy and Camille Vacher [Barguñó et al., 2010] (Section I.3.4) for a class strictly extending TAGED, in particular with arithmetic constraints (Section I.3.5, see also [Seidl et al., 2008]) and constraints similar to XML *unary key constraints*. This type of automata is well-suited to the analysis of XML schemes (problems of satisfiability, inclusion or equivalence of schemes) that are presented as the conjunction of a type constraint, which can be expressed in general by a tree automaton [Schwentick, 2007], and integrity constraints, which can be expressed by global constraints. Generalizing a previous construction [Filiot et al., 2008] we prove in [Barguñó et al., 2010] (Section I.3.6) a decision result for the satisfiability of an extension of the second-order monadic logic interpreted on trees, by reduction to the emptiness problem.

Another emptiness decision procedure for TAGED, with a better complexity than [Barguñó et al., 2010], is presented in the thesis of Camille Vacher [Vacher, 2010] (Section I.3.3). It works by reduction to the emptiness problem for automata calculating on the compression of trees in the form of directed acyclic graphs (DAGs). A work in progress with Anca Muscholl and Igor Walukiewicz also aims to extend this idea to more general classes of tree automata with global constraints.

Outline of Part II: Verification of Infinite State Systems

In the *regular model checking* approaches for verification, the transitions between the configurations of a system are classically represented by formalisms for the transformation of trees, such as transducers or rewrite systems. A crucial problem in this context is the finite characterization (by automata) of the closure of tree languages by these transformations.

We consider first the case of *term rewriting systems*, (the terms are labeled trees of bounded rank, Section II.1). We analyze in [Jacquemard et al., 2009, 2011a] the closure of the rigid tree automata languages under rewriting, with a decision algorithm for the membership of a given tree to this closure (Section II.1.4). In addition, we have solved by the negative the open problem of the decidability of *reachability* and *confluence* for flat term rewriting systems [Jacquemard, 2003; Mitsuhashi et al., 2006] and showed [Godoy and Jacquemard, 2009], using tree automata techniques, that the problem the *uniqueness of normal forms* (*i.e.* that a given rewrite system,

seen as a computation formalism, is functional) is decidable for flat and linear term rewriting systems and undecidable for flat and right-linear term rewriting systems (Section II.1.2).

We have also studied the closure of unranked ordered tree languages under rewrite systems for such trees, called *hedge rewriting systems* (Section II.2). We have shown [Jacquemard and Rusinowitch, 2008a] (Section II.2.2) that, on the one hand, rewrite systems called context-free (because their rules are shaped like productions of context-free tree grammars) transform languages of unranked ordered tree automata (called *hedge automata* [Murata, 1999]) into an extension called *context-free hedge automata*, and on the other hand, that the symmetric of these rewriting systems preserve the hedge automata languages. In addition, we proposed with Michael Rusinowitch [Jacquemard and Rusinowitch, 2010] (Section II.2.3) an extension of hedge rewriting systems with some parameters representing hedge automata languages. A parametrized hedge rewriting rule can represent an infinity of hedge rewriting rules. This provides a natural model for atomic operations for *updating* XML documents (renaming, insertion, deletion, replacement) as specified in the W3C recommendation of XQuery Update Facility 1.0 [Robie et al., 2011] (Section II.2.4). Some results of forward and backward type inference [Jacquemard and Rusinowitch, 2010] for arbitrary iterations of these operations (by construction of hedge automata and context-free hedge automata) allow us to verify properties of *consistency* of *access control policies* (for read and write access) for XML documents (Section II.2.5). The verification of security policies has also been the subject of several internships in bilateral projects with Tunisia and of an ARC INRIA [Youssef et al., 2009; Abbassi et al., 2010].

In Section II.3, we consider the closure of tree languages by application of rewriting with special *strategies*, a problem which has been less studied than for plain rewriting. With Andrea Gascon and Guillem Godoy we show [Gascón et al., 2008] (Section II.3.1) the surprising result that (in the case of terms) the application of the *innermost* strategy, which is the analogous to the call-by-value computations for functional languages, gives better results, *wrt* regular model checking, than plain rewriting [Jacquemard, 2003]: the closure of a regular tree language by a flat term rewriting system is a language of a well known tree automata model with local constraints, with good properties [Bogaert and Tison, 1992].

We have also studied with Masahiko Sakai and Yoshiharu Kojima [Jacquemard et al., 2011b], following an internship of the latter at LSV, the control of term rewriting by contextual conditions expressed by tree automata: computations of these automata are used to define the selection of the rewrite positions (Section II.3.2). This work was motivated by the analysis of XML read/write access control policies, because in practice, the positions of application of updates are usually selected by XPath expressions [Robie et al., 2011] and this selection can be defined by tree automata.

Finally, in Section II.3.3, we present some preliminary results obtained with Adrien Boiret and Luc Segoufin [Boiret, 2010] toward the study of context-free unranked ordered tree languages. The definition a sufficiently expressive and decidable class of context-free languages for unranked ordered trees is an interesting perspective for the representation of closures of unranked tree languages under hedge rewriting

with strategies, in the context of the analysis of XML transformations.

I

Classes of Extended Tree Automata

We present several classes of tree automata which strictly extend in expressiveness the standard tree automata, either on ranked or unranked trees. They are classified into three families, presented in the next three sections, and for each family, we have chosen one domain of application. Several models are presented for each family. Tough we do not claim an exhaustive covering, we do not restrict to the models that we have studied.

Standard Tree Automata

Ranked terms.

A *signature* Σ is a finite set of *function symbols* with arity. We sometimes denote Σ in extenso as $\{f_1 : a_1, \ldots, f_n : a_n\}$ where $f_1, \ldots, f_n$ are the function symbols, and the integers $a_1, \ldots, a_n$ are the corresponding arities. We denote the subset of function symbols of Σ of arity n as Σ_n. A signature is called *unary* if all its function symbols have arity 0 or 1. The set of *ranked terms* (or simply terms) over the signature Σ and a countable set of variables $\mathcal{X}$ is defined recursively as $\mathcal{T}(\Sigma, \mathcal{X}) := \mathcal{X} \cup \{f(t_1, \ldots, t_n) \mid n \geq 0, f \in \Sigma_n, t_1, \ldots, t_n \in \mathcal{T}(\Sigma)\}$. The set of variables of $\mathcal{X}$ occurring in a term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is denoted $vars(t)$. A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is called *ground* if $vars(t) = \emptyset$. The set of ground ranked terms is denoted by $\mathcal{T}(\Sigma)$. A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is called *linear* if every variable of $vars(t)$ occurs at most once in t.

The *positions* in terms are denoted by sequences of non-zero natural numbers; ε denotes the empty sequence (*root* position), and $p.p'$ denotes the concatenation of positions p and p'. The prefix ordering on positions is denoted $p \preceq p'$ and two positions p, p' incomparable with respect to this ordering are called *parallel*, denoted by $p \parallel p'$. The *domain* (set of positions) of a ranked term $t = f(t_1, \ldots, t_n)$ $(n \geq 0)$ is defined recursively as $\mathcal{P}os(t) = \{\varepsilon\} \cup \{i.p \mid 1 \leq i \leq n, p \in \mathcal{P}os(t_i)\}$. It is traditional to see a ranked term $t \in \mathcal{T}(\Sigma)$ as a function from its set of positions $\mathcal{P}os(t)$ into Σ. For this reason, the symbol labeling the position p in t shall be denoted by $t(p)$. The *height* of a term t, denoted by $h(t)$, is the maximal length of a position of $\mathcal{P}os(t)$. In particular, the length of ε is 0. A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is called *flat* if it is of height at most one, and *shallow* if every variable of $\mathcal{X}$ occurs at a height at most one in t.

The *subterm* of t at position p, denoted $t|_p$, is defined recursively by $t|_\varepsilon = t$ and $f(t_1, \ldots, t_n)|_{i.p} = t_i|_p$. The *replacement* in t of the subterm at position p by s, denoted $t[s]_p$, is defined recursively by $t[s]_\varepsilon = s$ and $f(t_1, \ldots, t_{i-1}, t_i, t_{i+1}, \ldots, t_n)[s]_{i.p} = f(t_1, \ldots, t_{i-1}, t_i[s]_p, t_{i+1}, \ldots, t_n)$.

A *substitution* is a mapping from a finite subset of $\mathcal{X}$ into $\mathcal{T}(\Sigma, \mathcal{X})$. The definition of a substitution σ can be extended homomorphically from variables to arbitrary terms by $\sigma(x) = x$ for all $x \in \mathcal{X} \setminus dom(\sigma)$ and $\sigma(f(t_1, \ldots, t_n)) = f\big(\sigma(t_1), \ldots, \sigma(t_n)\big)$. A *variable renaming* is a substitution from variables to variables. Given a substitution σ, the term $\sigma(t)$ is called an *instance* of the term t. A substitution σ is *grounding* for a term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is $\sigma(t)$ is ground.

A *context* of dimension n is a linear term $C \in \mathcal{T}(\Sigma, \{x_1, \ldots, x_n\})$. When not otherwise specified, we shall consider contexts of dimension 1. The context $C = x_1$ is called the *trivial context*. Given a context C of dimension n and n terms $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{X})$, we write $C[t_1, \ldots, t_n]$ to denote $\sigma(C)$ where σ is the substitution $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$.

Ranked Tree Automata.

A *ranked tree automaton* (TA for short) over a signature Σ is a tuple $\mathcal{A} = \langle \Sigma, Q, F, \Delta \rangle$ where Q is a finite set of *states*, $F \subset Q$ is the subset of final states and Δ is a set of *transition rules* of the form $f(q_1, \ldots, q_n) \to q$, with $f \in \Sigma_n$, $q_1, \ldots, q_n, q \in Q$. Sometimes, we shall refer to $\mathcal{A}$ as a subscript of its components, like in $Q_\mathcal{A}$ to indicate that Q is the state set of $\mathcal{A}$.

A *run* of a TA $\mathcal{A}$ over Σ on a ranked term $t \in \mathcal{T}(\Sigma)$ is a function r from $\mathcal{P}os(t)$ into $Q_\mathcal{A}$ such that for each $p \in \mathcal{P}os(t)$, $t(p)\big(r(p.1), \ldots, r(p.n)\big) \to r(p)$ is a transition rule of $\Delta_\mathcal{A}$ ($n \geq 0$). The run r is called *successful* (or *accepting*) if the state symbol $r(\varepsilon)$ at its root is in $F_\mathcal{A}$. By abuse of notation, we shall use the term notations (subterm, replacement...) for the runs. The *language* $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of terms $t \in \mathcal{T}(\Sigma)$ on which there exists a successful run of $\mathcal{A}$. For every state $q \in Q$, we denote $\mathcal{L}(\mathcal{A}, q)$ the language of $\mathcal{A}$ in state q, which is the set of terms $t \in \mathcal{T}(\Sigma)$ on which there exists a run r of $\mathcal{A}$ with $r(\varepsilon) = q$; Hence $\mathcal{L}(\mathcal{A}) = \bigcup_{q \in F} \mathcal{L}(\mathcal{A}, q)$. A set of ranked terms of $\mathcal{T}(\Sigma)$ is called *regular* if it is the language of some TA.

Equivalently, we can define recursively the languages $\mathcal{L}(\mathcal{A}, q)$ as the smallest subsets of $\mathcal{T}(\Sigma)$ such that

$$\mathcal{L}(\mathcal{A}, q) \subseteq \bigcup_{\substack{f(q_1, \ldots, q_n) \to q \in \Delta \\ n \geq 0}} \big\{ f(t_1, \ldots, t_n) \mid \forall i, 1 \leq i \leq n, t_i \in \mathcal{L}(\mathcal{A}, q_i) \big\},$$

or also as the least fixed points solutions in the Σ-algebra of terms of a set of equations $X_q = \bigcup_{f(q_1, \ldots, q_n) \to q \in \Delta} f(X_{q_1}, \ldots, X_{q_n})$ for each state $q \in Q$, see *e.g.* [Courcelle, 1989].

Example 1 *A very classical example of regular tree language is the set of Boolean expression which evaluate to true. Let* $\Sigma = \{\wedge : 2, \vee : 2, \neg : 1, \top, \bot : 0\}$*, and let*

$\mathcal{A} = \langle \Sigma, \{q_0, q_1\}, \{q_1\}, \Delta \rangle$, *where* Δ *contains the rules*

$$\begin{array}{rclrclrclrcl}
\bot & \to & q_0, & \top & \to & q_1, & & & & & & \\
\neg(q_0) & \to & q_1, & \neg(q_1) & \to & q_0, & & & & & & \\
\vee(q_0, q_0) & \to & q_0, & \vee(q_0, q_1) & \to & q_1, & \vee(q_1, q_0) & \to & q_1, & \vee(q_1, q_1) & \to & q_1, \\
\wedge(q_0, q_0) & \to & q_0, & \wedge(q_0, q_1) & \to & q_0, & \wedge(q_1, q_0) & \to & q_0, & \wedge(q_1, q_1) & \to & q_1
\end{array}$$

This TA *will evaluate every true Boolean expression in* $\mathcal{T}(\Sigma)$ *into* q_1 *and every false expression into* q_0*. For instance, the unique run of* $\mathcal{A}$ *on* $t = \wedge(\wedge(\top, \vee(\top, \neg(\bot))), \neg(\top))$ *is* $r = q_0(q_1(q_1, q_1(q_1, q_1(q_0))), q_0(q_1))$. ◇

Example 2 *The following* TA $\mathcal{A} = \langle \Sigma, \{q, q_\neg, q_\mathsf{f}\}, \{q_\mathsf{f}\}, \Delta \rangle$ *over the same signature as in Example 1, recognizes the terms containing the pattern* $\neg(\neg(x))$*. Its transition rules are*

$$\begin{array}{rclrclrclrcl}
\bot & \to & q, & \top & \to & q, & & & & & & \\
\neg(q) & \to & q, & \neg(q) & \to & q_\neg, & \neg(q_\neg) & \to & q_\mathsf{f}, & \neg(q_\mathsf{f}) & \to & q_\mathsf{f}, \\
\vee(q, q) & \to & q, & \wedge(q, q) & \to & q & & & & & & \\
\vee(q_\mathsf{f}, q_*) & \to & q_\mathsf{f}, & \vee(q_*, q_\mathsf{f}) & \to & q_\mathsf{f}, & \wedge(q_\mathsf{f}, q_*) & \to & q_\mathsf{f}, & \wedge(q_*, q_\mathsf{f}) & \to & q_\mathsf{f}
\end{array}$$

where $q_* \in \{q, q_\neg, q_\mathsf{f}\}$*. This* TA *can be very convenient for instance to recognize the terms which can simplified by an equation line* $\neg(\neg(x)) = x$. ◇

The construction of Example 2 is generalizable: given a linear term t, the set of ground terms over Σ containing an instance of t as a subterm (ground terms *embedding* t) is a regular tree language.

An operational semantics of ranked tree automata can be defined by the application of the transitions rules as (ranked) term rewriting rules (see definition in Section 1.5), where the state symbols are seen as new constant function symbols (of arity 0). With this approach (see *e.g.* [Comon et al., 2007]), a computation of a TA $\mathcal{A}$ on a ranked term t is a rewrite sequence, starting from t and reducing it, with the transition rules of $\mathcal{A}$, into a single state q (in this case, we have $t \in \mathcal{L}(\mathcal{A}, q)$). This defines *bottom-up* computations of ranked tree automata. An alternative is to consider *top-down* computations, when the rewriting applies the transition rules in the other direction (the top-down counterpart of the transition rule $f(q_1, \ldots, q_n) \to q$ is $q \to f(q_1, \ldots, q_n)$). A top-down computation start from a state q and generate a term of $\mathcal{L}(\mathcal{A}, q)$ (it is then a derivation of a regular ranked term grammar). This approach is convenient for dealing with infinite terms (which is out of the scope of this document) or for the definition of alternating tree automata.

It is possible to add to TA some ε-transitions of the form $q \xrightarrow{\varepsilon} q'$, where q and q' are states, without changing their expressiveness.

Determinism

A TA $\mathcal{A}$ over Σ is called *deterministic* (resp. *complete*) if for all $f \in \Sigma_n$ and all states $q_1, \ldots, q_n$ of $\mathcal{A}$, there is at most (resp. at least) one state q of $\mathcal{A}$ such that $\mathcal{A}$ contains

a transition $f(q_1, \ldots, q_n) \to q$. If $\mathcal{A}$ is deterministic (resp. complete) then for all tanked term $t \in \mathcal{T}(\Sigma)$, there exists at most (resp. at least) one run of $\mathcal{A}$ on t. Hence, a is deterministic and complete TA $\mathcal{A}$ over Σ can be seen as a function from $\mathcal{T}(\Sigma)$ to $Q_{\mathcal{A}}$, such that for all $t \in \mathcal{T}(\Sigma)$, $\mathcal{A}(t)$ is the unique state q such that $t \in \mathcal{L}(\mathcal{A}, q)$.

Similarly as for finite automata, for all TA $\mathcal{A}$ over Σ, one can built in PTIME a complete TA $\mathcal{A}_c$ whose size is polynomial in the size of $\mathcal{A}$, and such that $\mathcal{L}(\mathcal{A}_c) = \mathcal{L}(\mathcal{A})$. The construction works by addition of a *trash state*, accepting all terms. Also, for all TA $\mathcal{A}$ over Σ, one can built in EXPTIME with a *subset construction* a deterministic TA $\mathcal{A}_d$ whose size is exponential in the size of $\mathcal{A}$ (the sets of $\mathcal{A}_d$ are subsets of $Q_{\mathcal{A}}$) and such that $\mathcal{L}(\mathcal{A}_d) = \mathcal{L}(\mathcal{A})$. It can be shown that the exponential size for the determinization is a lower bound (it is in particular a consequence of the properties below).

If we consider the top-down semantics, then the definition of determinism (of $\mathcal{A}$ over Σ) varies: for all state q of $\mathcal{A}$ and $f \in \Sigma$, we want at most tuple of states $(q_1, \ldots, q_n)$ such that $q \to f(q_1, \ldots, q_n)$ is a top-down transition rule of $\mathcal{A}$. With this definition, not all TA can be determinized (consider for instance the regular language $\{f(a, b), f(b, a)\}$).

There is also an analogous of the Myhill-Nerode Theorem for TA, establishing the equivalence of the regularity of a ranked term language $L \subseteq \mathcal{T}(\Sigma)$ and the finite index of the congruence[1] $\equiv_L$ defined by $s \equiv_L t$ iff for all context C over Σ, $C[s] \in L \Leftrightarrow C[t] \in L$. Consequently, for every regular ranked term language $L \subseteq \mathcal{T}(\Sigma)$, there exists a *minimal* deterministic TA over Σ recognizing L, and its number of states is the index of $\equiv_L$.

Boolean Closures.

The class of regular ranked terms languages is closed under the Boolean operations, with some procedures of automata construction working within the following complexity bounds on time and size of the automata obtained[2]

union: linear (union of two tree automata, assuming that their the state sets are disjoint)

intersection: quadratic (Cartesian product of automata)

complementation: exponential (determinization, completion, and inversion of final and non-final states).

The exponential complexity for complementation is also a lower bound in general, for deterministic TA however, the construction for the complementation is polynomial.

[1] A *congruence* $\equiv$ on $\mathcal{T}(\Sigma)$ is an equivalence relation such that for all $f \in \Sigma_n$, if $s_1 \equiv t_1, \ldots, s_n \equiv t_n$, then $f(s_1, \ldots, s_n) \equiv f(t_1, \ldots, t_n)$.

[2] Without further precisions, we consider that the *size* of a TA (or an extension in the next sections) is its number of state symbols plus the sum of the respective sizes of its transition rules, *i.e.* it is the number of symbols used in the definition of the automaton.

Decision Problems.

The two central problems in automata theory in our context are *membership* and *emptiness* decision, which are defined as follows. We shall spend some time to describe decision algorithms for these two problems, as we shall study them for all automata models considered in this document.

Membership Problem:

given a TA $\mathcal{A}$ over Σ and a term $t \in \mathcal{T}(\Sigma)$, decide whether $t \in \mathcal{L}(\mathcal{A})$.

The membership problem is decidable in PTIME for TA. A polynomial decision algorithm for a TA $\mathcal{A} = \langle \Sigma, Q, F, \Delta \rangle$ and a term $t \in \mathcal{T}(\Sigma)$ works by computing a labeling function r from $\mathcal{P}os(t)$ into 2^Q, which is the unique run of $\mathcal{A}_d$ on t. Only the necessary states are computed, on the fly, in a bottom-up relabeling of t. The membership problem is actually LOGCFL-complete for general TA and its exact complexity is still unknown in the case of deterministic TA.

Emptiness Problem:

given a TA $\mathcal{A}$ over Σ, decide whether $\mathcal{L}(\mathcal{A}) = \emptyset$.

The emptiness problem is solved by a simple incremental algorithm marking every state q such that $\mathcal{L}(\mathcal{A}, q) \neq \emptyset$, by iteration of the following step: if q is not marked and there exists $f(q_1, \ldots, q_n) \to q \in \Delta$ such that $q_1, \ldots, q_n$ are marked, then mark q. Using appropriate data structures for $\Delta_{\mathcal{A}}$, the algorithm stops in linear time either when a final state is marked ($\mathcal{L}(\mathcal{A}) \neq \emptyset$) or if there are no more states to mark (and then $\mathcal{L}(\mathcal{A}) = \emptyset$). The correctness of this approach can be proved using a *pumping argument*: in a term $t \in \mathcal{L}(\mathcal{A})$, accepted by $\mathcal{A}$ with a successful run r, replacing a subterm of t at a position p by any term of $\mathcal{L}\big(\mathcal{A}, r(p)\big)$ returns a term which is still in $\mathcal{L}(\mathcal{A})$. In particular, doing replacements by smaller subterms *wrt* a well-founded ordering $<$, total on $\mathcal{T}(\Sigma)$, containing the strict subterm relation and monotonic *wrt* context application (see *e.g.* [Dershowitz and Jouannaud, 1990]) permits ones to run a minimality argument. Another way to decide emptiness in linear time is to reduce this problem to the satisfiability of the set of propositional Horn clauses $X_{q_1}, \ldots, X_{q_n} \Rightarrow X_q$ such that there exists $f(q_1, \ldots, q_n) \to q$ in Δ. The problem is PTIME-complete (see *e.g.* [Veanes, 1997]).

Finiteness Problem:

given a TA $\mathcal{A}$ over Σ, decide whether $\mathcal{L}(\mathcal{A})$ is finite.

This problem is decidable in PTIME, with an algorithm testing for the existence (in the TA transitions) of a loop from an non-empty state q to itself, such that a final state is reachable from q.

The three following problems, which are PSPACE-complete for finite automata, become EXPTIME-complete (equivalently, alternating PSPACE-complete) for tree automata.

Problem of the Emptiness of Intersection:

given n TA $\mathcal{A}_1, \ldots, \mathcal{A}_n$ over Σ, decide whether $\mathcal{L}(\mathcal{A}_1) \cap \ldots \cap \mathcal{L}(\mathcal{A}_n) = \emptyset$?

This problem is related to type inference in logic programming [Frühwirth et al., 1991], a proof of EXPTIME-hardness for non-deterministic and deterministic TA can be found in [Veanes, 1997] and for deterministic top-down TA in [Seidl, 1994].

Problem of Universality:

given a TA $\mathcal{A}$ over Σ, decide whether $\mathcal{L}(\mathcal{A}) = \mathcal{T}(\Sigma)$.

This problem is EXPTIME-complete. Universality is however decidable in PTIME for deterministic TA.

Problem of Inclusion:

given two TA $\mathcal{A}_1$ and $\mathcal{A}_2$ over Σ, decide whether $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$.

This problem, whose universality is a particular case, is EXPTIME-complete, and it is decidable in PTIME when $\mathcal{A}_2$ is deterministic.

A study of the parametrized complexity of the above problems has been recently conducted [Barecka and Charatonik, 2011] for several classes of tree automata. As a rule of thumb, the difficult problems remain (unfortunately) untractable even after fixing a parameter like for instance the number of states.

1 Tree Automata with Local Constraints

In this section, we consider some extensions of the classical tree automata model with the ability to perform some tests on the input term, at each computation step (a survey on theses classes can be found in the Chapter 4 of [Comon et al., 2007]). These formalisms have been applied for deciding some problems related to term rewriting and in deductive and inductive first order theorem proving. The models presented here are restricted to the recognition of ranked terms.

A severe limitation of standard tree automata is their inability to test for equality (isomorphism) or disequality between subterms in an input term. For instance, the pattern matching ability of TA, illustrated by Example 2, is strictly limited to *linear* pattern (without multiple occurrence of variables). The language of terms matching a non-linear pattern such as $\vee\big(x, \neg(x)\big)$ is not regular. This can be shown by contradiction, using a pumping argument as in the proof of the *emptiness problem* on page 17. In order to overcome this limitation and improve the expressive power of ranked tree automata, the extensions presented in the next sections have been proposed.

1.1 Local Equalities and Disequalities Constraints

A *ranked tree automaton with equality and disequality constraints* [Caron, 1993] (TAC for short) over a signature Σ is a tuple $\mathcal{A} = \langle \Sigma, Q, F, \Delta \rangle$ where Σ, Q and F are like for TA, and the transitions rules of Δ have the form $f(q_1, \ldots, q_n) \xrightarrow{c} q$, where $f \in \Sigma_n$, $q_1, \ldots, q_n, q \in Q$ and c is a Boolean combination without negation of equality and disequality constraints of the respective form $\pi = \pi'$ and $\pi \neq \pi'$, where π and π' are positions.

A *run* of a TAC $\mathcal{A}$ over Σ on a ranked term $t \in \mathcal{T}(\Sigma)$ is a function from $\mathcal{P}os(t)$ into $Q_\mathcal{A}$ such that for each $p \in \mathcal{P}os(t)$, there exists a a transition rule $t(p)\big(r(p.1), \ldots, r(p.n)\big) \xrightarrow{c} r(p)$ in $\Delta_\mathcal{A}$ such that $t|_p$ satisfies c, where the satisfaction of an atomic constraint $\pi = \pi'$ (resp. $\pi \neq \pi'$) by a term $s \in \mathcal{T}(\Sigma)$ is defined by $\pi, \pi \in \mathcal{P}os(t)$ and $s|_\pi = s|_{\pi'}$ (resp. $s|_\pi \neq s|_{\pi'}$). Successful runs and languages of TAC are defined the same way as for TA. Note that the transition rules of TAC with a trivial constraint equal to true have the same behaviour as TA transitions (hence TAC strictly extend TA). For the sake of simplicity, we shall write $f(q_1, \ldots, q_n) \to q$ instead of $f(q_1, \ldots, q_n) \xrightarrow{true} q$.

Example 3 *The following TAC* $\mathcal{A} = \big\langle \Sigma, \{q, q_\neg, q_\mathsf{f}\}, \{q_\mathsf{f}\}, \Delta \big\rangle$ *over the same signature as in Example 1, recognizes the terms matching the non-linear pattern* $\vee(x, \neg(x))$*, where* Δ *is the following set of transition rules:*

$$\begin{array}{l} \bot \to q, \quad \top \to q, \quad \neg(q) \to q, \qquad \neg(q) \to q_\neg, \\ \vee(q, q) \to q, \quad \wedge(q, q) \to q, \quad \vee(q, q_\neg) \xrightarrow{1=2.1} q_\mathsf{f} \end{array}$$

Note the constraint in the last transition rule, ensuring the equality of the subterms at the two positions of the variable x *in* $\vee(x, \neg(x))$*. In order to extend* $\mathcal{A}$ *to the recognition of ranked terms embedding the above pattern (i.e. ranked terms with a subterm matching this pattern) it is sufficient to add the following transition rules, ensuring the propagation up to the root position of the final state* q_f*:* $\neg(q_\mathsf{f}) \to q_\mathsf{f}$*,* $\vee(q_\mathsf{f}, q_*) \to q_\mathsf{f}$*,* $\vee(q_*, q_\mathsf{f}) \to q_\mathsf{f}$*,* $\wedge(q_\mathsf{f}, q_*) \to q_\mathsf{f}$*,* $\wedge(q_*, q_\mathsf{f}) \to q_\mathsf{f}$*, where* $q_* \in \{q, q_\neg, q_\mathsf{f}\}$*.* $\diamond$

The class of TAC languages is still closed under all Boolean operations. In particular, the TAC can be determinized and completed – a TAC $\mathcal{A}$ is called *deterministic* (resp. *complete*) if for all tanked term $t \in \mathcal{T}(\Sigma)$, there exists at most (resp. at least) one run of $\mathcal{A}$ on t.

Unfortunately, the emptiness problem is undecidable for TAC [Mongy, 1981; Bogaert, 1990], even when the constraints are restricted to conjunctions of equalities (like in the original model called RATEG of [Mongy, 1981]) and even when this equalities are between cousins positions [Tommasi, 1992] (*i.e.* they have the form $\pi = \pi'$ where π and π' have length 2). This can be shown by an encoding of the Post correspondence problem. The key idea for the encodings in these undecidability proofs is the possibility for the TAC to overlap some (local) equality tests. An overlap occurs when an equality $\pi = \pi'$ is tested at a position p by a transition rule, and another equality $\pi_0 = \pi'_0$ is tested at a position $p_0 \prec p$, such that $p \preceq p_0.\pi_0 \prec p.\pi$ and $p \preceq p_0.\pi'_0 \prec p.\pi'$. It appeared that this notion of *equality overlapping* is important for drawing the boundaries between decidable and undecidable classes of ranked tree automata with local constraints, as we will try to describe in the next sections.

Note that the membership problem is still decidable in PTIME for TAC, just by adding the (PTIME) verification of the constraints to the algorithm presented at page 17.

Brother Constraints

A decidable subclass of TAC called *ranked tree automaton with brother equality and disequality constraints* (TAB) has been proposed in [Bogaert and Tison, 1992]. Every TAB is a TAC in which every constraint $\pi = \pi'$ or $\pi \neq \pi'$ is such that both π and π' have length one. Hence, TAB are restricted to test equality and disequality between subterms at brother positions. Note that no equality overlap is possible in the computations of TAB.

Example 4 *The following TAB* $\mathcal{A} = \langle \Sigma, \{q, q_f\}, \{q_f\}, \Delta \rangle$ *over the signature* Σ *of Example 1, recognizes the terms matching the non-linear pattern* $\vee(x, x)$*, where* Δ *is the following set of transition rules:*

$$\begin{array}{ccc} \bot \to q, & \top \to q, & \neg(q) \to q \\ \vee(q,q) \to q, & \wedge(q,q) \to q, & \vee(q,q) \xrightarrow{1=2} q_f \end{array}$$

Note however that the language of Example 3 is not recognizable by a TAB.

The set of balanced binary terms built over the signature $\Sigma' = \{\bot : 0, \vee : 2\}$ *is recognized by the TAB* $\mathcal{B} = \langle \Sigma', \{q\}, \{q\}, \{\bot \to q, \vee(q,q) \xrightarrow{1=2} q\} \rangle$. ◇

The class of TAB languages is closed under all Boolean operations and the emptiness problem is decidable, and EXPTIME-complete, for TAB [Bogaert and Tison, 1992] (it is decidable in PTIME for deterministic TAB). In Section II.3.1, we present a result on the closure under rewriting with the innermost strategy which involves the construction of a TAB.

1.2 Reduction Automata

Another decidable subclass of TAC has been proposed in [Dauchet et al., 1995]. A *reduction automaton* (RA for short) is defined as a TAC $\mathcal{A} = \langle \Sigma, Q, F, \Delta \rangle$ together

with a partial ordering $<$ on Q such that for all transition rule $f(q_1, \ldots, q_n) \xrightarrow{c} q \in \Delta$, q is an upper bound (wrt $\leq$) of $\{q_1, \ldots, q_n\}$, and it is moreover a strict upper bound if c contains an equality $\pi = \pi'$. With this restriction, the number of equalities tested along a computation path in a run of $\mathcal{A}$ is bounded (by the number of states of $\mathcal{A}$), hence in particular the number of equality overlap is bounded as well.

Example 5 *The TAC of Example 3, recognizing the ranked terms embedding the non-linear pattern $\vee(x, \neg(x))$, it a RA, with the ordering $<$ defined by $q < q_\neg < q_\mathsf{f}$.* ◇

This example illustrates a remarkable property of RA, that for every ranked term t (with variables, possibly not linear), the set of ground ranked terms embedding t is recognizable by a deterministic and complete RA $\mathcal{A}_t$ whose size is polynomial in the size of t, and which is constructed in PTIME in the size of t. Moreover, the class of RA languages is closed under union and intersection and the subclass of deterministic and complete RA is closed under complementation. Also, the class of RA and TAB are incomparable in expressiveness.

In [Dauchet et al., 1995], it is shown that the emptiness problem is decidable for deterministic and complete RA. The decidability proof is based on a rather involved pumping argument. Indeed, the simple pumping argument presented in Section I for TA is no longer valid for TAC. Let $\mathcal{A}$ be a TAC and t a ranked term accepted by $\mathcal{A}$ with a run r. Even if some ranked term $t' < t$ is accepted in the state $r(p)$ (for some position $p \in \mathcal{P}os(t)$), the replacement $t[t']_p$ may no longer be accepted by $\mathcal{A}$. The reason is that some equality or disequality tested by $\mathcal{A}$ above p in t might be invalidated by the replacement (of $t|_p$ by t'). The principle of the proof of [Dauchet et al., 1995] is that if $t \in \mathcal{L}(\mathcal{A})$ is big enough, it is possible to find some pumpings (actually combinations of pumpings) in t which do not invalidate the equalities and disequalities tested by the constraints of $\mathcal{A}$ in the unique run r of $\mathcal{A}$ on t. The case of disequalities is complicated, and is described in the next section. The case of equalities is simpler, using parallel replacements on equivalence classes of subterms whose size is bounded because of the ordering condition on states in the definition of the RA.

For RA with the additional restriction that there is no overlapping between constraints, emptiness is decidable in PTIME.

It was thought [Dauchet et al., 1995] that emptiness is also decidable for non-deterministic RA. However we have shown several years later, with Michael Rusinowitch and Laurent Vigneron that it is not the case.

Theorem 1 [Jacquemard et al., 2006]

The emptiness problem is undecidable for RA.

We built a RA $\mathcal{A}$ which recognizes a set of ranked terms representing computations of a given 2-counter machine $\mathcal{M}$. A configuration of $\mathcal{M}$ is represented by a term $p(s^{n_1}(0), s^{n_2}(0))$ where p is the current state of $\mathcal{M}$ and $s^{n_1}(0)$, resp. $s^{n_2}(0)$, is

the representation of the content of the first, resp. second, counter, in unary using the symbols 0 and successor s. It is possible to express with a conjunction of equalities $\pi = \pi'$ that in a ranked term $g(c_{i+1}, g(c_i, 0))$, c_i and c_{i+1} are successive configurations of $\mathcal{M}$. However, iterating such tests in a sequence of configurations $g(c_n, g(c_{n-1}, \dots g(c_1, g(c_0, 0))))$ would require an unbounded number n of overlapping equality tests along the rightmost branch (on the g positions), and this is not possible with RA.

We circumvent this problem using redundancy in the representation of computations of $\mathcal{M}$, transitivity of subterm equality and non-determinism of RA. The above sequence of successive configurations is represented by the term t_n, defined recursively by $t_0 = h(g(c_0, 0), 0)$ and $t_{i+1} = h(g(c_{i+1}, t_i), t_i)$. All equality tests for successive configurations will be performed by $\mathcal{A}$ on the g positions, which are all parallel in t_n. Moreover, for testing the equality between the two occurrences of t_i in t_{i+1}, it is sufficient to test $1.2 = 2$ at the root of t_n and $2.2 = 2.1.2$ at every g position. It is possible for $\mathcal{A}$ to realize all these equality tests, while fulfilling the RA condition on the states. However, this requires to have different runs of $\mathcal{A}$ on the two occurrences of t_i in t_{i+1} (one run will test the equalities, and the other not), *i.e.* $\mathcal{A}$ must be non-deterministic.

Consequently, the RA cannot be determinized. It is not known whether the class of RA languages is closed under complementation or not.

1.3 Restriction to Disequality Constraints

The complexity of the emptiness decision procedure of [Dauchet et al., 1995] for RA is rather high (a tower of several exponentials in time) and the exact complexity of the problem is unknown. In an old work with Hubert Comon [Comon and Jacquemard, 1997, 2003], we focus on another subclass of TAC which was sufficient for our purpose (the decision of *ground reducibility*, see Section 1.5 below). In this subclass, that we denote by $\mathsf{TAC}_{\neq}$, the constraints in transition rules are conjunctions of disequalities $\pi \neq \pi'$, and we managed to reduce the time complexity of emptiness decision to one exponential.

Theorem 2 [Comon and Jacquemard, 1997, 2003]

Emptiness is decidable in EXPTIME for $\mathsf{TAC}_{\neq}$.

More precisely, the complexity upper bound of our emptiness decision algorithm, for a $\mathsf{TAC}_{\neq}$ $\mathcal{A} = \langle \Sigma, Q, F, \Delta \rangle$, is $O\big((|Q| \times |\Delta|)^{P(\mathcal{A})}\big)$, where P is a polynomial in the size of the constraints in the transition rules of $\mathcal{A}$. The proof follows the same principles (combined pumping argument) as the proof for RA, but with some new combinatorial arguments in order to reduce the complexity upper bound, in particular a state marking algorithm based on the pumping argument, where several marks per state are needed. This algorithm avoids the enumeration of all ranked terms smaller that the bound above which a pumping is possible.

An important difference with [Dauchet et al., 1995] is that we consider general replacement for the pumping, whereas in [Dauchet et al., 1995], a pumping is restricted to be the replacement of $t|_p$ (for $p \in \mathcal{P}os(t)$) by one of its subterms. Finding pumping which do not invalidate disequalities tested by the automaton is done following the same principle as in [Dauchet et al., 1995] though. This involves on one hand an enumeration for bounding the cases of disequality test invalidated by pumping close to the test position, and on the other hand, the trick that if some disequality is invalidated by a pumping not too close to the test position, then it means that some subterms of t are equal and we can combine replacements in this subterms in parallel.

No Overlap

Another decidable model of tree automata with local constraints has been introduced more recently [Godoy et al., 2010], in order to solve the problem of deciding whether the image of a regular (ranked) tree language under a given homomorphism is regular. This model called $\mathsf{TA}_{hom,\neq}$ in [Godoy et al., 2010] is not (up to my knowledge) a subcase of TAC, but it is closely related. The automata contain arbitrary disequality constraints defined as above, and limited equality constraints, which, roughly, cannot be superposed.

It is also shown in [Godoy et al., 2010] that the complement of a $\mathsf{TAC}_{=}$ (the subclass of TAC whose constraints are conjunctions of atomic equality constraints) is a $\mathsf{TAC}_{\neq}$, and vice-versa (both constructions are exponential). We will come back in Part II to an important consequences of this result.

1.4 Deciding Regularity

The decidability of *regularity* (whether the language of a given tree automaton with constraint is regular) is in general difficult to establish for strict extensions of TA. In this section, we propose a general criteria for proving its undecidability.

It is well know that it is undecidable whether a given CF language is regular. This is a consequence of a more general Greibach's Theorem ([Hopcroft and Ullman, 1979], § 8.7) which states that this undecidability result holds for every class of languages satisfying some closure properties and for which universality is undecidable. This result can be generalized to ranked tree languages. The *composition* of $L_1 \subseteq \mathcal{T}(\Sigma_1)$,..., $L_n \subseteq \mathcal{T}(\Sigma_n)$ by a n-ary symbol d (possibly in one of the signatures Σ_i) is $d(L_1, \ldots, L_n) = \{d(t_1, \ldots, t_n) \mid t_1 \in L_1, \ldots, t_n \in L_n\}$.

Theorem 3

Let Γ be a class of tree recognizers whose set of corresponding languages is strictly larger than the set of regular tree languages, and stable under union and composition. Then universality for Γ is reducible to regularity for Γ.

With Luis Barguñó, Carlos Creus, Guillem Godoy, and Camille Vacher, we show in [Barguñó et al., 2010] this result for a particular tree automata model presented in Section 3. However, the proof principle is very general and can be applied to Theorem 3 as well. Let N be a non-regular language of Γ over a signature Σ, and let d be a new function symbol with arity 2, not in Σ. Let $\mathcal{C} \in \Gamma$, over the same signature Σ, be an input for the problem of universality, and let $\mathcal{C}' \in \Gamma$ be such that $\mathcal{L}(\mathcal{C}') = d\big(\mathcal{T}(\Sigma), N\big) \cup d\big(\mathcal{L}(\mathcal{C}), \mathcal{T}(\Sigma)\big)$.

It holds that $\mathcal{L}(\mathcal{C}) = \mathcal{T}(\Sigma)$ if and only if $\mathcal{L}(\mathcal{C}')$ is regular. If $\mathcal{L}(\mathcal{C}) = \mathcal{T}(\Sigma)$, then $\mathcal{L}(\mathcal{C}') = d\big(\mathcal{T}(\Sigma), \mathcal{T}(\Sigma)\big)$, and it is regular. For the other direction, assume that $\mathcal{L}(\mathcal{C}) \neq \mathcal{T}(\Sigma)$ and let $s \in \mathcal{T}(\Sigma) \setminus \mathcal{L}(\mathcal{C})$. By construction, $N = \{t \mid d(s,t) \in \mathcal{L}(\mathcal{C}')\}$. It follows that $\mathcal{L}(\mathcal{C}')$ is not regular.

Stability under union and composition is a reasonable requirement, satisfied by all the classes of tree automata previously presented. It is not difficult to show that universality is undecidable for RA and $\mathsf{TA}_{\neq}$, for instance by constructing an automaton recognizing all the ranked terms which are not representing a computation of a given 2-counter machine. It can be defined as a union of automata, corresponding to the different error cases, where each case can be handled either by a TA or with some disequality constraints. It is also a consequence of the result of [Godoy et al., 2010] on the complementation of $\mathsf{TAC}_{=}$ into $\mathsf{TAC}_{\neq}$, and the undecidability of emptiness of $\mathsf{TAC}_{=}$. Then, following Theorem 3, we have the following result.

Corollary 4

Regularity is undecidable for RA and $\mathsf{TA}_{\neq}$.

Note that on the other hand, regularity of the set of irreducible ground terms for a given rewrite system is decidable [Vágvölgyi and Gilleron, 1992; Kucherov and Tajine, 1995] (these sets are languages of $\mathsf{TA}_{\neq}$, see below). Also, universality is decidable for $\mathsf{TA}_{=}$ [Godoy et al., 2010], hence Theorem 3 is not applicable to these automata. It is however easy to obtain a analogous result of Theorem 3 by reduction of the emptiness problem.

Theorem 5

Let Γ be a class of tree recognizers whose set of corresponding languages is strictly larger than the set of regular tree languages, and stable under composition. Then emptiness for Γ is reducible to regularity for Γ.

Let N be a non-regular language of Γ over a signature Σ, and let d be a function symbol of arity 2. Let $\mathcal{C} \in \Gamma$ be an input for the emptiness problem, over the same signature Σ, and let $\mathcal{C}' \in \Gamma$ be such that $\mathcal{L}(\mathcal{C}') = d\big(\mathcal{L}(\mathcal{C}), N\big)$. It holds that $\mathcal{L}(\mathcal{C}) = \emptyset$ if and only if $\mathcal{L}(\mathcal{C}')$ is regular. If $\mathcal{L}(\mathcal{C}) = \emptyset$, then $\mathcal{L}(\mathcal{C}') = \emptyset$, and it is regular. For the other direction, assume that $\mathcal{L}(\mathcal{C}) \neq \emptyset$. It is not difficult to show that the regularity of $\mathcal{L}(\mathcal{C}')$ would imply the regularity of N, hence $\mathcal{L}(\mathcal{C}')$ is not regular.

Corollary 6

Regularity is undecidable for $\mathsf{TA}_{=}$.

It is shown in [Bogaert et al., 1999] that regularity is decidable for TAB.

1.5 Applications to First Order Theorem Proving

Since many years, ranked tree automata have served as decision tool in the theory of term rewriting. In Part II of this document, we present several studies of the rewrite closure of tree languages, and applications to the verification of infinite state systems. Let us present in this section some applications of tree automata techniques to automated deduction.

Term Rewriting Systems

Term rewriting is a rule-based formalism for describing computations in ranked terms [Dershowitz and Jouannaud, 1990; Baader and Nipkow, 1998]. A *term rewriting system* (TRS) over a signature Σ is a finite set of rewrite rules $\ell \to r$, where $\ell \in \mathcal{T}(\Sigma, \mathcal{X}) \setminus \mathcal{X}$ (ℓ is called left-hand side (*lhs*) of the rule, and describes a pattern to be replaced in a term) and $r \in \mathcal{T}(\Sigma, vars(\ell))$ (r is called right-hand side (*rhs*) of the rule, and it is the new term for replacement).

A term $s \in \mathcal{T}(\Sigma, \mathcal{X})$ rewrites to t by a TRS $\mathcal{R}$ at a position p of s with a substitution σ, denoted $s \xrightarrow[\mathcal{R},p,\sigma]{} t$ (p and σ may be omitted in this notation) if there is a rewrite rule $\ell \to r \in \mathcal{R}$ such that $s|_p = \sigma(\ell)$ and $t = s[\sigma(r)]_p$. In this case, s is said to be *reducible*. The set of irreducible terms, also called *$\mathcal{R}$-normal forms*, is denoted by $\mathsf{NF}_{\mathcal{R}}$. The reflexive, transitive closure and reflexive, symmetric, transitive closure of the relation $\xrightarrow[\mathcal{R}]{}$ are denoted respectively $\xrightarrow[\mathcal{R}]{*}$ and $\xleftrightarrow[\mathcal{R}]{*}$. A term t' is a *normal form* of a term t for a TRS $\mathcal{R}$ if $t \xrightarrow[\mathcal{R}]{*} t'$ and $t' \in \mathsf{NF}_{\mathcal{R}}$.

A TRS $\mathcal{R}$ is *terminating* if there is no infinite chain of terms s_i, $i \geq 0$, such that $s_i \xrightarrow[\mathcal{R}]{} s_{i+1}$. A TRS $\mathcal{R}$ is *confluent* if for all terms t, s_1, s_2 such that $s_1 \xleftarrow[\mathcal{R}]{*} t \xrightarrow[\mathcal{R}]{*} s_2$ there exists a term t' such that $s_1 \xrightarrow[\mathcal{R}]{*} t' \xleftarrow[\mathcal{R}]{*} s_2$. A TRS is *convergent* if it is both terminating and confluent.

First Order Logics of Rewrite Relations

Tree automata techniques have been used to established results of decision of satisfiability for several first order logics related to rewriting. This is achieved by a compilation of formulas into tree automata and then decision of emptiness.

This technique works for instance for the encompassment theory, interpreted on the domain of ground terms $\mathcal{T}(\Sigma)$. The formulas of this theory are built with a finite number of unary predicates of the form E_t, for $t \in \mathcal{T}(\Sigma, \mathcal{X})$, such that $E_t(u)$ holds for $u \in \mathcal{T}(\Sigma)$ if t matches a subterm of u (u embeds t). The formulas of this logic can be compiled into deterministic reduction automata (RA) characterizing the

models, hence satisfiability is decidable [Dauchet et al., 1995]. Similar techniques apply also to the first order theory of a reduction relation, with a binary predicate interpreted as the rewrite relation $\xrightarrow[\mathcal{R}]{*}$, on ground terms, for a fixed TRS $\mathcal{R}$. Compilation into tree automata, hence decision of satisfiability, is possible when $\mathcal{R}$ is ground or $\mathcal{R}$ is such that for every rule $\ell \to r$, ℓ and r are linear and do not share variables [Dauchet and Tison, 1990]. These results can be used to decide some properties of TRS expressible in these logics.

In a recent work with Etienne Lozes, Ralf Treinen and Jules Villard [Jacquemard et al., 2011c], we generalize this approach for showing decidability of first order theories with several unary predicates interpreted by membership of ground terms to fixed regular tree languages, and several binary predicates interpreted as different congruence relations $\xleftrightarrow[\mathcal{R}_1]{*}$, ..., $\xleftrightarrow[\mathcal{R}_k]{*}$, where the TRS $\mathcal{R}_1$, ..., $\mathcal{R}_k$ satisfy the above conditions (*i.e.* they are linear and variable disjoint).

We show also that either relaxing the conditions on the TRS's or adding ternary predicates of the form $x = f(y, z)$ (for $f \in \Sigma$) leads to undecidability (even for ground TRS in the second case). The decidability result is applied to the model-checking problem of $A\pi\mathcal{L}$, a spatial equational logic for the applied π-calculus, using a reduction of this problem to the validity of first-order formulas in $\mathcal{T}(\Sigma)$ with multiple congruence relations.

Inductive Theorem Proving

The goal of inductive theorem proving is to automatize the proof of statements in some particular structures, like natural numbers, integers, lists, binary trees... Let us focus on specifications defined by finite sets of *equational Horn clauses* (universally quantified disjunctions of equations or negation of equations between terms). An equational clause C is an *inductive theorem* of such a specification $\mathcal{E}$, assumed consistent, if C is valid in the smallest Herbrand model of $\mathcal{E}$. A Herbrand model $\mathcal{H}$ of a set of first order sentences with equality has for domain a quotient of the set of terms $\mathcal{T}(\Sigma)$ by a congruence, and interprets every function symbol $f \in \Sigma$ as a term constructor (for $t_1, \ldots, t_n \in \mathcal{T}(\Sigma)$, $f(t_1, \ldots, t_n)$ is interpreted in $\mathcal{H}$ as the equivalence class of the term $f(t_1, \ldots, t_n)$). A set of first order formulas is consistent iff it has an Herbrand model, and in the case of Horn clauses, there exists a smallest Herbrand model, obtained by intersection of all the Herbrand models. It is also classically obtained as the least fixpoint of the immediate consequence operator defined by the Horn clauses.

Example 6 *The following set of equations specifies the addition of natural numbers* $\mathcal{E} = \{0 + x = x, s(x) + y = s(x + y)\}$. *The minimal Herbrand model of* $\mathcal{E}$ *is the quotient of the term algebra* $\mathcal{T}(\Sigma)$, *where* $\Sigma = \{0, s, +\}$, *by the equational of* $\mathcal{E}$. *It is isomorphic to the set of natural numbers, i.e. the set of terms* $\{0, s(0), s(s(0)), \ldots\} = \mathcal{T}\big(\{0, s\}\big)$. *The conjecture* $x + 0 = x$ *holds in this model. This can be shown using the classical induction scheme over natural numbers (*$0 + 0 = 0$ *holds, and assuming* $s^n(0) + 0 = s^n(0)$, *we can show* $s^{n+1}(0) + 0 = s^{n+1}(0)$ *using simplification with the*

equations of $\mathcal{E}$). Note however that $x + 0 = x$ is not a logical consequence of $\mathcal{E}$: there are some models of $\mathcal{E}$ (less "natural" than $\mathcal{T}(\{0, s\})$) in which it does not hold. ◇

Normal Form Automata

For every TRS $\mathcal{R}$ over a signature Σ, the set of $\mathcal{R}$-normal forms is the language of a tree automaton with disequality constraints ($\mathsf{TA}_{\neq}$ of Section 1.3). For several reasons explained in the two subsections below, this finite representation of normal forms is very interesting in inductive theorem proving. Intuitively, under some conditions, it is a finite presentation of the smallest Herbrand model.

Theorem 7 [Comon and Jacquemard, 2003]

For all TRS $\mathcal{R}$, there exists a $\mathsf{TA}_{\neq}$ $\mathcal{A}_{\mathsf{NF}(\mathcal{R})}$ whose size is exponential in the size of $\mathcal{R}$, and such that $L(\mathcal{A}_{\mathsf{NF}(\mathcal{R})})$ is the set of ground normal forms of $\mathcal{R}$.

The construction of $\mathcal{A}_{\mathsf{NF}(\mathcal{R})}$ is presented in Figure 1. Intuitively, it corresponds to the complementation and completion of a $\mathsf{TA}_{=}$ recognizing $\mathcal{R}$-reducible terms by pattern matching of left members of rewrite rules of $\mathcal{R}$, where every subset of states (for the complementation) is represented by the most general common instance (*mgi*) of its elements (if they are unifiable). The complementation of $\mathsf{TA}_{=}$ into $\mathsf{TA}_{\neq}$ is formally defined in [Godoy et al., 2010]. In Figure 1, we use the notion of the *linearized* version $lin(\ell)$ of a term $\ell \in \mathcal{T}(\Sigma, \mathcal{X})$, which is a term such that $\mathcal{P}os(lin(\ell)) = \mathcal{P}os(\ell)$, $lin(\ell)$ is linear and for all $p \in \mathcal{P}os(\ell)$ such that $\ell(p) \notin \mathcal{X}$, $lin(\ell)(p) = \ell(p)$. This term $lin(\ell)$ is unique up to variable renaming.

Figure 1 Construction of a $\mathsf{TA}_{\neq}$ $\mathcal{A}_{\mathsf{NF}}(\mathcal{R})$ recognizing the $\mathcal{R}$-normal forms of a TRS $\mathcal{R}$.

$$
\begin{array}{rcl}
\mathcal{L}(\mathcal{R}) & = & \{u \mid u \text{ is a strict subterm of } lin(\ell) \text{ for some } \ell \to r \in \mathcal{R}\} \\
\mathcal{A}_{\mathsf{NF}(\mathcal{R})} & = & \langle \Sigma, Q_{\mathsf{NF}}(\mathcal{R}), Q_{\mathsf{NF}}(\mathcal{R}), \Delta_{\mathsf{NF}}(\mathcal{R}) \rangle \text{ with} \\
Q_{\mathsf{NF}}(\mathcal{R}) & = & \left\{ \begin{array}{l} \langle mgi(t_1, \ldots, t_n) \rangle \mid \{t_1, \ldots, t_n\} \text{ is a maximal} \\ \quad \text{subset of } \mathcal{L}(\mathcal{R}) \text{ s.t. } t_1, \ldots, t_n \text{ are unifiable} \end{array} \right\} \\
\Delta_{\mathsf{NF}}(\mathcal{R}) & \multicolumn{2}{l}{\text{contains every } f(\langle u_1 \rangle, \ldots, \langle u_n \rangle) \xrightarrow{c} \langle t \rangle \text{ such that } f \in \Sigma,} \\
 & \multicolumn{2}{l}{f(u_1, \ldots, u_n) \text{ is not matched by a linear } \textit{lhs} \text{ of } \mathcal{R},} \\
 & \multicolumn{2}{l}{t \text{ is the } mgi \text{ of } \{u \mid \langle u \rangle \in Q_{\mathsf{NF}}(\mathcal{R}) \text{ and } u \text{ matches } f(u_1, \ldots, u_n)\}, \text{ and}} \\
 & \multicolumn{2}{l}{c = \bigwedge_{\substack{\ell \to r \in \mathcal{R}, \\ \ell \text{ matches } f(u_1, \ldots, u_n)}} \ \bigvee_{\substack{x \in vars(\ell) \\ \ell|_\pi = \ell|_{\pi'} = x, \pi \neq \pi'}} \pi \neq \pi'.}
\end{array}
$$

Example 7 *To take a somewhat trivial example, if $\mathcal{R}$ is obtained from the specification of the addition of natural numbers from Example 6 by orienting the 2 equations from left to right (into rewrite rules) we have $\mathcal{L}(\mathcal{R}) = \{0, x, s(x)\}$, $Q_{\mathsf{NF}}(\mathcal{R}) =$*

$\{\langle 0\rangle, \langle s(x)\rangle\}$, *and* $\Delta_{\mathsf{NF}}(\mathcal{R}) = \{s(\langle 0\rangle) \to \langle s(x)\rangle, s(\langle s(x)\rangle) \to \langle s(x)\rangle\}$. *The language of* $\mathcal{A}_{\mathsf{NF}(\mathcal{R})}$ *is then* $\mathcal{T}\big(\{0, s\}\big)$. ◇

Example 8 *Let us extend the specification of Examples 6, 7 into the following specification of relative integers:*

$$\mathcal{R} = \{0+x \to x, s(x)+y \to s(x+y), p(x)+y \to p(x+y), s(p(x)) \to x, p(s(x)) \to x\}.$$

With the construction of Figure 1, $\mathcal{L}(\mathcal{R}) = \{0, x, s(x), p(x)\}$, $Q_{\mathsf{NF}}(\mathcal{R}) = \{\langle 0\rangle, \langle s(x)\rangle, \langle p(x)\rangle\}$, *and* $\Delta_{\mathsf{NF}}(\mathcal{R}) = \{s(\langle 0\rangle) \to \langle s(x)\rangle, s(\langle s(x)\rangle) \to \langle s(x)\rangle p(\langle 0\rangle) \to \langle p(x)\rangle, p(\langle p(x)\rangle) \to \langle p(x)\rangle\}$. *The language of* $\mathcal{A}_{\mathsf{NF}(\mathcal{R})}$ *is then* $\mathcal{T}\big(\{0, s\}\big) \cup \mathcal{T}\big(\{0, p\}\big)$. ◇

Proof by Consistency and Ground Reducibility

Some approaches for automatic inductive theorem proving, in the case of equational specifications, work by trying to derive of an inconsistency from a set of equations or a TRS, and a conjecture, using first order deduction techniques [Huet and Hullot, 1982; Jouannaud and Kounalis, 1986; Kapur and Musser, 1987]. What characterizes these approaches is that the proofs do not use explicit induction schemes. They are therefore also called *inductionless induction*, see [Comon-Lundh, 2001] for a general framework for such techniques.

An example is *inductive completion* [Fribourg, 1989; Bachmair, 1991] when the specification is presented by a ground convergent TRS $\mathcal{R}$. This method is a restriction of the Knuth-Bendix completion procedure, applied to $\mathcal{R} \cup \{s = t\}$. An inconsistency in this case is the derivation of a rewrite rule $\ell \to r$ such that ℓ is not *ground reducible wrt* $\mathcal{R}$. It means that the intersection between the set of ground instances of ℓ and the ground $\mathcal{R}$-normal forms is empty. Intuitively, this inconsistency condition means that the initial model of $\mathcal{R}$ (set of ground $\mathcal{R}$-normal forms) is different from the initial model of $\mathcal{R} \cup \{s = t\}$. Therefore, the above construction of $\mathcal{A}_{\mathsf{NF}(\mathcal{R})}$ can be used as a decision tool for ground reducibility in an inductive completion procedure, with optimal complexity.

Theorem 8 [Comon and Jacquemard, 2003]

Ground reducibility is EXPTIME-complete.

This approach can be generalized to sets of Horn clauses with equations, using saturation by superposition calculi, see [Ganzinger and Stuber, 1993].

Implicit Inductive Theorem Proving

Another family of methods for inductive theorem proving uses some explicitly induction schemes in the proofs (unlike proof by consistency methods), but these schemes are not manually provided by the user but rather automatically computed by the system [Zhang et al., 1988; Bouhoula and Rusinowitch, 1995; Bouhoula, 1997]. Roughly,

these methods compute, in a preliminary step, a finite set of terms, called *cover-set* in [Zhang et al., 1988] and *test-set* in [Bouhoula and Rusinowitch, 1995; Bouhoula, 1997], which is used in the induction steps of the proof to instantiate induction variables.

It is common in inductive theorem proving to assume that the specifications are built with *constructor function symbols* (for constructing terms representing data) and *defined symbols* (representing the operations defined on constructor terms).

Example 9 *In Example 6, the symbols* 0 *and* s *are the constructors, and the symbol* $+$*, which occurs at the head of lhs of equations, is defined.* ◇

In the specification of Example 9, the constructors are called *free* because there are no equations between terms made only of constructor symbols. This is also the case of the simple specification of append on lists proposed in Introduction (where the constructors symbols are cons and nil). In the specification of Example 8, the constructors 0, s and p are not free, because of the rules $s(p(x)) \to x$ and $p(s(x)) \to x$.

The *sufficient completeness* of a TRS $\mathcal{R}$ is a condition expressing that every ground term is reducible to a constructor term using the rules of $\mathcal{R}$. If a TRS $\mathcal{R}$ with constructors is sufficiently complete and terminating, then a set of representatives for the smallest Herbrand model of $\mathcal{R}$ is the set of ground constructor normal forms of $\mathcal{R}$. When the constructors are *free* in $\mathcal{R}$, the set of ground constructor normal forms is simply the set of ground constructors terms and it is very easy in this case to define an induction schema. This situation is therefore convenient for inductive reasoning, and many inductive theorem provers require free constructors, termination and sufficient completeness.

However, the assumption of constructor-freeness is too restrictive to define complex data structures such as sets, sorted lists, powerlists, binary search trees, etc. Such data structure generate complex induction schemes, and the automation of inductive proofs is therefore difficult in this settings. The soundness of cover-set [Zhang et al., 1988] and test-set [Bouhoula and Rusinowitch, 1995; Bouhoula, 1997] induction techniques do not require that the constructors are free. But, when they are not free, cover-sets and test-sets are over-approximating induction schemes, in the sense that they may represent some reducible ground terms. This may cause the failure (a "don't know" result) of the induction proofs. Moreover, the refutational completeness of test-set induction technique is not guaranteed in this case.

Some progress has been done in [Bouhoula and Jouannaud, 2001] in the direction of handling specification with non-free constructors. It has been generalized in [Bouhoula et al., 2000] to membership equational logic. These approaches work by transforming the initial specification in order to get rid of rewrite rules between constructors terms, using the construction of normal form tree automata (I implemented this procedure as a module of the system Maude [Clavel et al., 1999]). The axioms for constructors are however assumed to be *left-linear* rewrite rules, which is still too restrictive for the specification of structures like sets or sorted lists...

With Adel Bouhoula [Bouhoula and Jacquemard, 2008], we propose a framework for inductive theorem proving in theories containing constrained rewrite rules between constructor terms (where the constraints express *e.g.* syntactic equality, disequality, ordering, membership in a fixed regular tree language) and constrained equational Horn clauses (more exactly, *constrained and conditional rewrite rules*) for defined functions. The key idea is a strong and natural integration of tree automata with constraints in an implicit induction procedure, where they are used as induction schema. The tree automata recognize *exactly* the sets of constructor normal forms, and are computed using a generalization of the procedure of Figure 1 to arbitrary symbolic constraints on the variables in *lhs* of rewrite rules (instead of equalities between this variables in the case of Figure 1). Therefore, the normal form tree automata constructed contains arbitrary constraints (intuitively, negations of the constraints of the rewrite rules between constructors). They are used in the procedure as induction schemes (for the generation of subgoals at induction steps, using the transition rules backward, like production rules of tree grammars) and, moreover, for several decision procedures similar to the consistency in the above proof by consistency, by reduction to emptiness test.

Our procedure is sound and refutationaly complete under the assumption of sufficient completeness of the given set of rewrite rules $\mathcal{R}$ and separate termination of the respective sets of rules for defined functions and for the constructors (there is no requirement for *termination* of the whole set of rules unlike [Bouhoula and Rusinowitch, 1995; Bouhoula, 1997]). It is moreover sound for disproofs (if the procedure fails, then the conjecture is not an inductive theorem), provided that $\mathcal{R}$ is *strongly* complete (a stronger condition for sufficient completeness) and ground confluent.

Moreover, the use of constraints permits us in some cases to use the constrained completion technique of [Kirchner et al., 1990] in order to transform a non-terminating theory into a terminating one, by the addition of ordering constraints in constructor rules. This allows in particular to make proofs modulo non orientable axioms, without having to modify the core of our procedure. Ordering constraints are also useful for specifying complex data structures. For instance in [Bouhoula and Jacquemard, 2008] we study a specification of *ordered lists*, build on the top of the example of lists (with nil and cons) in Introduction with new constrained rewrite rules such as $x_1 > x_2 \Rightarrow \mathsf{cons}(x_1, \mathsf{cons}(x_2, y)) \to \mathsf{cons}(x_2, \mathsf{cons}(x_1, y))$.

Joinable Inductive Theorem Proving

We have also proposed with Adel Bouhoula an adaptation of the above induction procedure [Bouhoula and Jacquemard, 2007] to relax the assumption of confluence of the specification. This approach is used for simultaneous proof of reachability properties and detection of attacks for finite set of processes communicating on an insecure network (see Section 2.7 below for more details on this setting). The non-confluence is needed in order to specify the non-deterministic behavior of the insecure network.

The specification language described above, combining conditional rewrite rules and constraints, enables to express concisely and independently the instructions of the processes, the behavior of the insecure network, the algebraic laws followed by the operators (the latter are specified as rules between constructors), and security properties of several kind, following a generic scheme based on a definition of traces.

Decision of Sufficient Completeness

We propose moreover with Adel Bouhoula, [Bouhoula and Jacquemard, 2006] and [Bouhoula and Jacquemard, 2011] a procedure for checking the sufficient completeness of specifications as above (conditional and constrained TRS). This procedure is integrated into the above framework for inductive theorem proving. The procedure is presented by an inference system which is shown sound and complete if the specification in input is convergent for ground terms. The goal of the system is the incremental construction of a finite *pattern tree* for each non-constructor symbol. The pattern trees are labeled by constrained terms and every construction step consists in the replacement of variables by terms, using backward the transitions of the constructor normal form automaton. Intuitively, the idea is to build a finite representation of all the terms of the form $f(t_1, \ldots, t_n)$ such that f is a defined symbol and $t_1, \ldots, t_n$ are ground constructor terms irreducible by $\mathcal{R}$ (if $\mathcal{R}$ is ground convergent, then it is sufficiently complete iff every such term is reducible). We show that it is sufficient in these settings to consider a finite set of positions of variables to be replaced, and therefore that the construction terminates.

A precondition of one inference of this system refers to a (undecidable) property called *strong ground reducibility*. This sufficient condition for ground reducibility requires in particular that the conditions of candidate rules of $\mathcal{R}$ (for reducing ground instances) are inductive consequences of $\mathcal{R}$, hence it is undecidable in general for conditional term rewriting systems, this condition is discharged to the above inductive theorem proving system. We show that strong ground reducibility and hence sufficient completeness become decidable for unconditional (*i.e.* purely equational) TRS which may contain constraints.

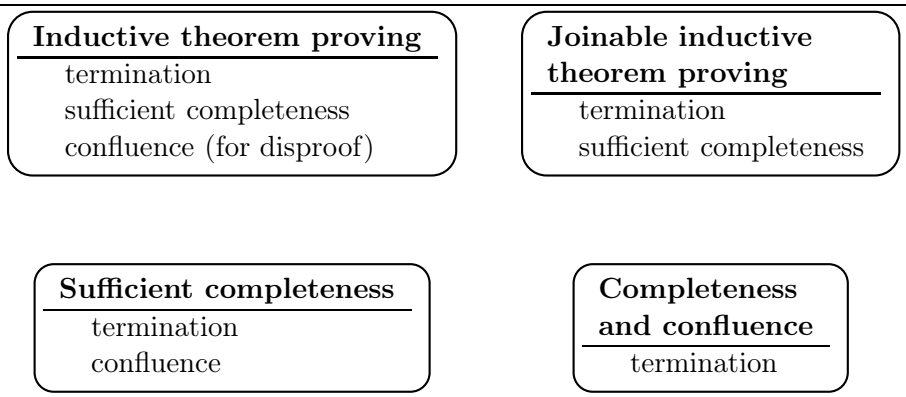
We have successfully applied our method to several examples, returning readable proofs and, in case of negative answer, a counter-example suggesting how to complete the specification.

An ongoing work, still with Adel Bouhoula, is concerned with the simultaneous decision of completeness and confluence for specifications as above. Finally, Figure 2 summarizes the joint works presented above with the assumptions.

2 Tree Automata as Sets of Horn Clauses

We consider in this section another presentation of tree automata with local constraints, based on a definition as finite sets of Horn clauses, following an approach initiated by Frühwirth et al [Frühwirth et al., 1991], for type inference in logic programming. This approach has had many developments, in particular in the con-

Figure 2 Procedures for conditional constrained TRS, with assumptions.



text of verification of infinite states systems [Nielson et al., 2001; Verma et al., 2005; Seidl and Verma, 2007, 2008; Verma and Goubault-Larrecq, 2007b; Seidl and Verma, 2009]. In Sections 2.1-2.4 we present several tree automata models and decision results, summarizing a joint work with Michael Rusinowitch and Laurent Vigneron [Jacquemard et al., 2006] and [Jacquemard et al., 2008a], in a uniform framework:

- the automata are defined as Horn clauses sets (this presentation is also adopted in Sections 2.5 and 2.6),
- the decision procedures are based on the saturation of clauses sets under a paramodulation calculus,
- we consider one unified decision problem, expressible in this settings, and generalizing most of the problems previously mentioned.

A Horn clause is a disjunctions of literals $-A_1 \vee \ldots \vee -A_k \vee H$, (every literal is an atom A or the negation of an atom $-A$) denoted $A_1, \ldots, A_k \Rightarrow H$, where $-A_1, \ldots, -A_k$ are negative literals called *antecedents* (the set of antecedents is also called the *body* of the clause) and H is a positive literal called *head* of the clause. When H is missing, $A_1, \ldots, A_k \Rightarrow$ is called a *goal* clause, and otherwise it is called *definite*. A definite clause without negative literals is called *positive*. We call *clausal tree automaton* a finite sets of definite Horn clauses built over a signature Σ, a finite set unary predicates denoted by lowercase letter p, q,..., and the equality predicate $=$.

Every predicate symbol other than $=$ occurring in a given clausal tree automaton $\mathcal{A}$ is called a *state* of $\mathcal{A}$. Given a clausal tree automaton $\mathcal{A}$ and a state q of $\mathcal{A}$, the language of $\mathcal{A}$ in q, denoted by $\mathcal{L}(\mathcal{A}, q)$, is the set of terms $t \in \mathcal{T}(\Sigma)$ such that $q(t)$ is a logical consequence of $\mathcal{A}$, *i.e.* such that $q(t)$ is true in the smallest (*wrt* inclusion) Herbrand model of $\mathcal{A}$ (note that such a model always exists and is unique since all the clauses of $\mathcal{A}$ are definite).
The *unified problem* mentioned above is the following

Problem of Membership to the Intersection of Instances (MII):

given a clausal tree automaton $\mathcal{A}$ over Σ, $n \geq 1$ states of $\mathcal{A}$, $q_1, \ldots, q_n$ and n terms $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, X)$, decide whether there exists a substitution σ, grounding for $t_1, \ldots, t_n$ and such that $\sigma(t_i) \in \mathcal{L}(\mathcal{A}, q_i)$ for all $1 \leq i \leq n$.

We will see below that MII generalizes many decision problems presented in the previous pages. Following the above definitions, the problem MII is satisfied by $\mathcal{A}$, $q_1, \ldots, q_n$ and $t_1, \ldots, t_n$ iff the set of clauses $\mathcal{A} \cup \{q_1(t_1), \ldots, q_n(t_n) \Rightarrow\}$ is unsatisfiable. The above fact suggests the use of classical *first order theorem proving* techniques [Bachmair and Ganzinger, 2001; Nieuwenhuis and Rubio, 2001] in order to solve decisions problems like MII, following for instance the approaches [Verma, 2003], [Goubault-Larrecq, 2005]. The main originality of our approach is the use of paramodulation, an *equational* calculus (instead *e.g.* of resolution in [Goubault-Larrecq, 2005]). This permits us to consider two extensions of standard tree automata which are important in the context of system verification: tree automata languages modulo equational theories (Section 2.2) and tree automata with local equality constraints (Section 2.3), as well as the combination of both (Section 2.4). The idea is that, in this framework, both the equational theories and the equality constraints can be represented directly in the clausal formalism.

We consider with Michael Rusinowitch and Laurent Vigneron [Jacquemard et al., 2008a] a calculus called *basic ordered paramodulation with selection* [Bachmair et al., 1995; Nieuwenhuis and Rubio, 2001]. It is sound (every inference rule of the calculus generate from 2 clauses ϕ_1 and ϕ_2 in premise a new clause ϕ which is a logical consequence of ϕ_1 and ϕ_2) and refutationally complete (from every unsatisfiable set of clauses, the inference system will generate, with a fair strategy, the empty clause). Hence, in order to solve the MII problem for a given clausal tree automaton $\mathcal{A}$, it is sufficient to try to generate the empty clause by iteration of the rules of this paramodulation calculus, starting from the clauses of $\mathcal{A}$ and a goal clause of the form $q_1(t_1), \ldots, q_n(t_n) \Rightarrow$. Decidability is obtained for a clausal model of tree automata as soon as it can be established that the iteration will terminate for automata of this class, *i.e.* starting from the clauses of an automaton and a goal clause as above, at some point, the set of clauses will be *saturated* (no new clause can be generated). The decidability results of the next Sections 2.1 and 2.3 are obtained by showing such termination results, either by showing that all the clauses which will be generated are of some type containing only a finite number of clauses (this number gives a complexity bound in the worst case), or by exhibiting a well-founded ordering such that a generated clause is strictly smaller than its premises.

We won't present in detail the inference rules of the calculus of basic ordered paramodulation with selection (there are given in Figure 3 for the sake of completeness, one may refer to [Jacquemard et al., 2008a] for more details). Instead, we will just describe briefly some characteristics, in particular the control which permits us to obtain the termination results. The paramodulation calculus works by application

Figure 3 Basic ordered paramodulation with selection. The conditions missing above are: (iii) $\ell\sigma \not\preceq r\sigma$ and $\ell\sigma = r\sigma$ is strictly maximal in $\Gamma\sigma, \ell\sigma = r\sigma$, (v) $u\sigma = v\sigma$ is maximal in $\Gamma'\sigma, u\sigma = v\sigma$, where σ is the most general unifier (mgu) of $\theta, \theta', \ell' = \ell, x = r$, (v') $u\sigma = v\sigma$ is maximal in $\Gamma'\sigma, u\sigma = v\sigma, A\sigma$ (σ is as in (v)).

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

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

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

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

$$\frac{\Gamma, s = t \Rightarrow A \; [\![\theta]\!]}{\Gamma \Rightarrow A \; [\![\theta, s = t]\!]} \text{ Equation Solving}$$

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

$$\frac{B, \Gamma \Rightarrow H[\![\theta]\!]}{B \Rightarrow q_B[\![\theta]\!] \quad q_B, \Gamma \Rightarrow H[\![\theta]\!]} \; \varepsilon\text{-splitting}$$

where the literals of $\Gamma \cup H$ are not equational, $B\theta$ is a set of literals of the form $q_1(x), \ldots, q_n(x)$, x is a variable which does not occur in Γ and H, and where q_B is a nullary uniquely associated with B, modulo variable renaming.

of equational reasoning *i.e.* replacement of equals by equals. Therefore, it will only involve literals that are equations. This is not a real limitation since non-equational atoms of the form $q(t)$ can be represented as equations $q(t) = \mathit{true}$, where *true* is a distinguished nullary function symbol in Σ. Therefore, below, by abuse, we shall talk about terms to refer either to first order terms or to literals.

The *basic strategy* requires that no inference will be performed in parts of terms which were generated by unification in previous steps. In practice, this is realized with constrained clauses, where the constraint (denoted in brackets in Figure 3) stores equations representing the result of unifications, and no inference is performed on the term in the constraint.

The principle of the *ordered strategy* and *selection* are the following. The inference rules are parametrized by a reduction ordering $\succ$ and a selection function. A *reduction ordering* is a well-founded ordering on atoms, stable under application of substitutions and contexts), assumed total on ground atoms. In [Jacquemard et al., 2008a], it is defined as an extension of a precedence ordering defined on the symbols of Σ and the predicates, to a lexicographic path ordering [Dershowitz and Jouannaud, 1990]. A *selection function* assigns to each clause a set of selected negative literals. The

strategy, roughly, restricts the inference rules to take place only on selected literals or literals which are either maximal in a clause (and will be reduced).

Furthermore, we use a rule called *ε-splitting* [Goubault-Larrecq, 2005], [Goubault-Larrecq et al., 2005], a variant of *splitting without backtracking* [Riazanov and Voronkov, 2001]. The principle of this rule is to replace a clause by two split clauses (the other rules of the paramodulation calculus add new clauses). Each of the split clauses contains a new nullary predicate q_B corresponding to a set of literals of the form $q_1(x), \ldots, q_n(x)$ (*ε-block*). Roughly, this nullary predicate is true as soon as the intersection of the languages of $q_1, \ldots, q_n$ is not empty. More formally, one of the split clauses is $q_1(x), \ldots, q_n(x) \Rightarrow q_B$, and the other is obtained by replacement of the ε-block by q_B, in the body of the clause, if the variable x does not occur elsewhere in the clause. With a careful choice of the ordering $\succ$, the effect of the rule of *ε-splitting* will be to force, before the treatment of a clause C, a pre-treatment to decide the non-emptiness of the intersection of $q_1, \ldots, q_n$. This is important in order to achieve termination. Note that the number of nullary literals add by this rule is bounded.

2.1 Automatic Clauses

We consider first the clauses called *automatic*, of the form

$$q_1(x_1), \ldots, q_n(x_n) \Rightarrow q\big(f(x_1, \ldots, x_n)\big) \qquad \text{(aut)}$$

where $n \geq 0$ and $x_1, \ldots, x_n$ are distinct variables.

It is easy to see that the class of languages of clausal tree automata made only of automatic clauses coincide with the regular tree languages: a clause as above corresponds indeed to a bottom-up TA transition of the form $f(q_1, \ldots, q_n) \to q$. Hence, let us call, by abuse, *tree automaton* (TA) every finite set of automatic clauses.

It was observed in [Goubault-Larrecq, 2005] that a resolution calculus (whose rules can be simulated by the above paramodulation calculus), with the strategies presented above (ordered strategy, selection and ε-splitting) permits ones to solve MII, by saturation. This results also holds for the above paramodulation calculus.

Theorem 9 [Goubault-Larrecq, 2005; Jacquemard et al., 2008a]

Ordered paramodulation with selection and ε-splitting terminates in a number of steps at most exponential, when performed on the union of a TA $\mathcal{A}$ and a goal clause $q_1(t_1), \ldots, q_n(t_n) \Rightarrow$.

Let us discuss the complexity of the problem of membership to the intersection of instances (MII) in the case of TA. When $n = 1$, we have a problem sometimes called *membership of an instance*. It is known to be EXPTIME-complete in general [Tison, 2000]. When $n = 1$ and t_1 is a ground term, then the problem is equivalent to a *membership* problem for $\mathcal{A}$, and when t_1 is a variable, then it is equivalent to a *non-emptiness* problem for $\mathcal{A}$. Hence it is solvable in PTIME in the two latter

cases (see page 17). The problem is also solvable in PTIME when $n = 1$ and the term t_1 is linear, and it is NP-complete when $n = 1$ and $\mathcal{A}$ is deterministic (and t_1 arbitrary), see *e.g.* [Comon et al., 2007]. When $t_1 = \ldots = t_n = x$ (a variable), MII is equivalent to the problem of non-emptiness of intersection of tree automata, which is EXPTIME-complete (page 18).

The termination result of Theorem 9 permits us to obtain the EXPTIME upper-bound for MII in the case of TA (this result is probably folklore knowledge).

Corollary 10

The problem MII is EXPTIME-complete for TA.

Extending the TA with *facts* of the form $\Rightarrow q(t)$ does not increase their expressiveness when q is a state and t is a ground or linear term. However, this is not the case when t can be non-linear.

Theorem 11 [Jacquemard et al., 2008a]

The problem of membership of an instance is undecidable for the union of a TA and facts.

2.2 Automatic Clauses Modulo an Equational Theory

Combining tree automata and term rewriting techniques for the verification of infinite state systems is the object of Part II. A problem in this context is to extend the decidability results on tree automata languages to equivalence classes of terms modulo an equational theory. Some authors, *e.g.* [Verma and Goubault-Larrecq, 2007a; Ohsaki and Takai, 2002a], have investigated the problem of emptiness decision for tree automata modulo specific equational theories, *e.g.* A, AC, ACU... Moreover, it is also shown in [Ohsaki and Takai, 2002a] that emptiness is decidable for regular languages modulo any linear equational theory.

In [Jacquemard et al., 2006], we propose with Michael Rusinowitch and Laurent Vigneron to apply the approach presented in the above paragraph to this problem. Indeed, an advantage of the clausal formalism is that we can add the equations of an equational theory directly in the set of clauses defining an automaton. Moreover, the above basic paramodulation calculus is still able to deal with such set of equational (positive) clauses, without preprocessing like *e.g.* in [Blanchet, 2001].

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

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

A *tree automaton modulo an equational theory* (TAE) is a finite set of clauses of type (aut) and (eq).

In [Jacquemard et al., 2008a], we consider conditions on equational theories called $\succ$-*convergence* and monadicness which ensures the termination of the above basic paramodulation calculus on TAE. An equational theory is called $\succ$-*convergent* if every equation $\ell = r$ is orientable by the ordering $\succ$, *i.e.* $\ell \succ r$, and the rewrite relation induced is confluent. The orientation ensures that the ordered paramodulation will always replace ℓ by r, and the confluence ensures that no paramodulation inferences can take place between the equations of the theory. An oriented equation $\ell = r$, with $\ell \succ r$, is *monadic* if r is either a variable occurring in ℓ or a term of the form $g(z_1, \ldots, z_k)$ for some $g \in \Sigma$, $k \geq 0$ and some distinct variables $z_1, \ldots, z_k$ occurring in ℓ.

Example 10 *The following axiom for integer equality:* $\mathsf{eq}(\mathsf{s}(x), \mathsf{s}(y)) = \mathsf{eq}(x, y)$ *as well as this axiom for the elimination of stuttering in lists:* $\mathsf{cons}(x, \mathsf{cons}(x, y)) = \mathsf{cons}(x, y)$ *are monadic (oriented) equations.* ◇

Theorem 12 [Jacquemard et al., 2008a]

Basic ordered paramodulation with selection and ε-splitting terminates when performed on the union of a TAE $\mathcal{A}$ modulo a $\succ$-convergent and monadic equational theory and a goal clause $p_1(t_1), \ldots, p_n(t_n) \Rightarrow$.

The saturation of the clauses of Theorem 12 roughly works in two steps (the order of the steps is guarantied by the ordered and selection strategies): first, superposition of the equational clauses into the automatic clauses, producing two-ways like clauses, and second, treating these two-ways clauses as in [Goubault-Larrecq, 2006]. The proof of termination of Theorem 12 is based on an invariant on the kind of clauses produced, which all belong to a finite type of clauses. The use of a basic strategy is crucial for this termination result.

Corollary 13 [Jacquemard et al., 2008a]

The problem MII is decidable for TA modulo a $\succ$-convergent monadic equational theories.

2.3 Automata Clauses with Equality Constraints

It is also possible to use literals built with the equality predicate $=$ in the body of clauses, in order to represent local equality constraints. In [Jacquemard et al., 2008a], we apply with Michael Rusinowitch and Laurent Vigneron this idea to the definition of a tree automata model with equality constraints generalizing those of reduction automata [Dauchet et al., 1995] (see Section 1.2), with equality tests between arbitrary terms. Note that the clausal tree automata with equality constraints

of [Jacquemard et al., 2008a] do not contain disequality constraints, unlike reduction automata.

Following the idea of [Dauchet et al., 1995], the clausal transitions with equality constraints are restricted in order to obtain decidability. Actually, we needed to reinforce the conditions of [Dauchet et al., 1995] in [Jacquemard et al., 2008a], in order to avoid the undecidability problem of Theorem 1. For this purpose, we use the precedence ordering (total on the function and predicate symbols) that is used for the definition of the reduction ordering $\succ$ on literals. Let us denote $>$ the precedence ordering.

Let $\mathcal{P}_0 \uplus \mathcal{P}_1$ be a partition of the set of unary predicates (states) into *ordinary predicates* of $\mathcal{P}_0$ and *test predicates* of $\mathcal{P}_1$. We shall sometimes mark a predicate q like in $\hat{q}$ to indicate that it is a test predicate. The constrained transitions of our automata have the following form:

$$q_1(x_1), \ldots, q_n(x_n), u_1 = v_1, \ldots, u_k = v_k \Rightarrow \hat{q}(x) \qquad (\mathsf{deep})$$

where $n, k \geq 0$, $x_1, \ldots, x_n, x$ are distinct variables, $u_1, v_1, \ldots, u_k, v_k \in \mathcal{T}\big(\Sigma, \{x_1, \ldots, x_n, x\}\big)$, $q_1, \ldots, q_n, q \in \mathcal{P}_0 \uplus \mathcal{P}_1$, $\hat{q}$ is a test predicate, and for all $i \leq n$, if q_i is a test predicate then $\hat{q} > q_i$.

The unconstrained transitions are restricted automatic clauses of type (aut) which contain no more test predicates symbols in their antecedents than in their heads.

$$q_1(x_1), \ldots, q_n(x_n) \Rightarrow q\big(f(x_1, \ldots, x_n)\big) \qquad (\mathsf{aut}')$$

where $n \geq 0$, $x_1, \ldots, x_n$ are distinct variables, and either $q_1, \ldots, q_n, q \in \mathcal{P}_0$ or q is a test predicate and at most one of $q_1, \ldots, q_n$ is equal to q, and the others belong to $\mathcal{P}_0$.

A *tree automaton with equational constraints* (TAD) is a finite set of clauses of type (aut′) or (deep). Note that every TA is a particular case of TAD (without test predicates and with all states in $\mathcal{P}_0$).

Example 11 *The language of the following TAD in state q_2 is the set of stuttering lists of natural numbers build with the symbols* cons *and* empty*:*

$$\begin{array}{rcl@{\qquad}rcl}
\Rightarrow & q_0(0) & & q_0(x_1) & \Rightarrow & q_0(\mathsf{s}(x_1)) \\
\Rightarrow & q_1(\mathsf{empty}) & & q_0(x_1), q_1(x_2) & \Rightarrow & q_1(\mathsf{cons}(x_1, x_2)) \\
 & & & q_0(x_1), q_2(x_2) & \Rightarrow & q_2(\mathsf{cons}(x_1, x_2)) \\
 & & q_0(x_1), q_1(x_2), x_2 = \mathsf{cons}(x_1, y), x = \mathsf{cons}(x_1, x_2) & \Rightarrow & q_2(x)
\end{array}$$

◇

Theorem 14 [Jacquemard et al., 2008a]

Ordered paramodulation with selection and ε-splitting terminates when performed on the union of a TAD $\mathcal{A}$ and a goal clause $q_1(t_1), \ldots, q_n(t_n) \Rightarrow$.

The proof of termination is a long case analysis [Jacquemard et al., 2008a] showing that, starting with $\mathcal{A} \cup \{q_1(t_1), \ldots, q_n(t_n) \Rightarrow\}$, every step of ordered paramodulation *wrt* the ordering $\succ$ and an ad-hoc selection function, with eager ε-splitting, returns either a clause smaller than all its premises (*wrt* to a well founded ordering $\gg$) or a clause of a type containing only a finite number of clauses.

Two key points ensure this result. First, the selection function selects in priority the equality constraints (*e.g.* $u_i = v_i$ in a clause (deep)). Hence, equations in the bodies of clauses will be eliminated first, before these clauses can be involved in resolution. Second, when all such equations have been eliminated, the predicates in the clauses obtained satisfy the same ordering condition as for (deep). Hence, thanks to the ordering conditions on predicates, the application of such clauses in resolution makes clauses smaller *wrt* $\gg$.

2.4 Automata Clauses with equality constraints modulo equational theories

The extensions of tree automata with local equality constraints and of tree automata modulo equational theories (or more generally, the rewrite closure of regular tree languages, see Part II) have been well studied separately. However, to our knowledge, there are very few decision results for the combination of both extensions. One difficulty stems from the fact that local equality constraints which test, by definition, syntactic equalities (isomorphisms between subterms), do not combine well with languages terms modulo. This problem is discussed furthermore in Part II.

A solution to this problem is to generalize the constraints to equality modulo. The Horn clauses formalism is particularly well-suited for this purpose: it is sufficient to combine the clauses of Sections 2.2 (eq) and 2.3 (types (deep) and (aut′)) in order to obtain the automata expected. In the next paragraphs, we present two works following this approach.

TA with brother constraints modulo flat equational theories

We had observed already in a somewhat old work with Christoph Meyer and Christoph Weidenbach [Jacquemard et al., 1998] that the equality tests between brother positions à la [Bogaert and Tison, 1992], presented in Section 1.1, can be easily simulated in the Horn clauses representation of tree automata. Equations are not necessary for this purpose, since multiple occurrences of a variable suffice. More precisely, an automatic clause with equality constraints between brothers is a clause of the form

$$q_1(x_1), \ldots, q_n(x_n) \Rightarrow q\big(f(x_1, \ldots, x_n)\big) \qquad \text{(ab)}$$

where $n \geq 0$ and $x_1, \ldots, x_n$ are variables (not necessarily distinct).

In [Jacquemard et al., 1998], we show a saturation result under sorted superposition, a variant of the above calculus, for clauses of the following type, which is more general than (ab), modulo flat equational theories

$$p_1(t), \ldots, p_m(t), q_1(x_1), \ldots, q_n(x_n) \Rightarrow H \qquad \text{(fb)}$$

where $m, n \geq 0$, t is a flat term, $x_1,\ldots,x_n$ are variables (not necessarily distinct) and H is either an atom of the form $q(s)$, where s is a flat term, or an equation $\ell = r$ where ℓ and r are flat, or H is missing.

Theorem 15 [Jacquemard et al., 1998]

Sorted superposition terminates when performed on a finite set of clauses of type (fb).

It follows in particular that emptiness and emptiness of intersection are decidable for TAB with equality constraints only, modulo equational theories, and unification modulo theories (fb) is decidable. This latter result is extended in [Jacquemard et al., 1998] to so-called *semi-linear* theories, which are shown to be finitely transformable into sets of clauses of type (fb).

TA with general equational constraints modulo equational theories

A *tree automaton with equational constraints modulo an equational theory* (TADE, defined with Michael Rusinowitch and Laurent Vigneron [Jacquemard et al., 2006]) is the union of an equational theory (finite set of clauses of type (eq)) and of a TAD. Some combinations of clauses presented in the above sections are summarized in Table 1.

	no clauses (eq) (empty eq. theory)	clauses (eq)
std TA (aut)	TA	TAE
TA with constraints (deep)+(aut′)	TAD	TADE

Table 1: Summary of the automata classes of Sections 2.1-2.4.

In order to obtain a saturation result for TADE, we restrict in [Jacquemard et al., 2008a] the equational theories in consideration to be $\succ$-convergent (a condition presented in Section 2.2, and moreover to contain only *sublinear* and *collapsing* equations, which means that the equations have the form: $f(t_1,\ldots,t_m) = x$ where the subterms $t_1,\ldots,t_m$ are linear ($f(t_1,\ldots,t_m)$ may be non-linear) and x is a variable.

Example 12 *The following theory of projections on pairs is sublinear and collapsing:* $\mathsf{fst}(\mathsf{pair}(x,y)) = x$, $\mathsf{snd}(\mathsf{pair}(x,y)) = y$, *as well as the following equations on function on lists* $\mathsf{car}(\mathsf{cons}(x,y)) = x$, $\mathsf{cdr}(\mathsf{cons}(x,y)) = y$, $\mathsf{cons}(\mathsf{car}(x),\mathsf{cdr}(x)) = x$.

Other classical examples of sublinear and collapsing equations can specify some algebraic properties of cryptographic functions, like decryption in a symmetric cryptosystem $\mathsf{dec}\big(\mathsf{enc}(x,y),y\big) = x$ *(the symbols* enc *and* dec *stand for encryption and*

decryption and the variables x and y correspond respectively to the encrypted plaintext and the encryption key), or, in the case of public (asymmetric) key cryptography: $\mathsf{adec}(\mathsf{aenc}(x, \mathsf{pub}(y)), \mathsf{inv}(\mathsf{pub}(y))) = x$ *and* $\mathsf{adec}(\mathsf{aenc}(x, \mathsf{inv}(\mathsf{pub}(y))), \mathsf{pub}(y)) = x$ *where* inv *is an idempotent operator, following the axiom* $\mathsf{inv}(\mathsf{inv}(y)) = y$*, which associates to a public encryption key its corresponding private key (for decryption), and conversely.* ◇

Theorem 16 [Jacquemard et al., 2008a]

Basic ordered paramodulation with selection and ε-splitting terminates when performed on the union of TADE $\mathcal{A}$ modulo a $\succ$-convergent sublinear and collapsing equational theory and a goal clause $q_1(t_1), \ldots, q_n(t_n) \Rightarrow$.

The proof follows the same schema as for Theorem 14 (TAD). It is however much more involved, with a refinement of the ordering $\gg$ on clauses in order to take into account the number of occurrences of variables in positive literals (whereas, roughly, the multisets of predicates and of negative literals were sufficient for Theorem 14).

Implementation

A prototype has been developed [Jacquemard et al., 2008b], implementing the procedure of Theorem 16. The Ocaml sources are available at http://tace.gforge.inria.fr/.

Relating RA and TADE

We show in [Jacquemard et al., 2008a] that every reduction automaton containing only equality constraints is equivalent to a TADE of the same size, as long as its transitions fulfill the restrictions on predicates introduced in the definition of (aut′) and (deep) in order to preserve decidability of emptiness.

The idea is to use a $\succ$-convergent sublinear–collapsing theory defining projection symbols $\pi_1, \ldots, \pi_a$ of arity 1, where a is the maximal arity of a function symbol of Σ, with equational clauses of the form $\Rightarrow \pi_i(f(x_1, \ldots, x_n)) = x_i$ for all $f \in \Sigma_n$, $i \leq n$. Then, an equality constraint of the form $p = p'$, with $p = p_1 \ldots p_k$ and $p' = p'_1 \ldots p'_l$, with $k, l \geq 1$, is replaced by an equation $\pi_{p_2}(\ldots \pi_{p_k}(x_{p_1})) = \pi_{p'_2}(\ldots \pi_{p'_l}(x_{p'_1}))$ in a clause of type (deep).

Perspective: combination of TACE with brother equality constraints

The combination of the classes of TA with brother constrains of [Bogaert and Tison, 1992] (Section 1.1) and of reduction automata of [Dauchet et al., 1995] (Section 1.2) preserves emptiness decidability [Caron et al., 1994].

Hence it could be interesting to study in the clausal framework the combination of equality tests restricted by ordering conditions on states and unrestricted tests

between brother positions, possibly modulo equational theories, *i.e.* to study saturation procedures for the combination of clauses of type (deep), equational clauses (eq), and clauses of a type similar to (aut′) except that the variables $x_1, \ldots, x_n$ are not supposed to be distinct.

2.5 Pushdown and Visibly Pushdown Tree Automata

CF Tree Grammars and Pushdown tree automata

A *tree grammar* (see *e.g.* [Comon et al., 2007]) is a tuple $\mathcal{G} = \langle \mathcal{N}, S, \Sigma, P \rangle$ where $\mathcal{N}$ is a finite set of *non-terminal* symbols with arities, denoted by uppercase letters, $S \in \mathcal{N}$ has arity 0 (it is called the *axiom* of $\mathcal{G}$), Σ is a signature disjoint from $\mathcal{N}$ (its elements are also called *terminal* symbols), and P is a set of *production rules*, of the form $\ell \to r$ where ℓ, r are terms of $\mathcal{T}(\Sigma \cup \mathcal{N}, \mathcal{X})$ and such that ℓ contains at least one non-terminal. The derivation relation $t \xrightarrow[\mathcal{G}]{*} s$ induced by a tree grammar $\mathcal{G}$ is the reflexive and transitive closure of the rewrite relation defined by P. The *language* of the tree grammar $\mathcal{G}$ is defined as $\mathcal{L}(\mathcal{G}) = \{t \in \mathcal{T}(\Sigma) \mid S \xrightarrow[P]{*} t\}$.

A *context-free tree grammar* (CFTG) is a tree grammar whose production rules all have the form $X(x_1, \ldots, x_n) \to r$ where $X \in \mathcal{N}$ has arity n, $x_1, \ldots, x_n$ are distinct variables, and $r \in \mathcal{T}\big(\Sigma \cup \mathcal{N}, \{x_1, \ldots, x_n\}\big)$. When $r = x_i$ for some $i \leq n$, then the rule is called *erasing* or *collapsing*.

An notion of pushdown tree automata equivalent to CF tree grammars has been proposed in [Guessarian, 1983; Schimpf and Gallier, 1985; Coquidé et al., 1994], extending tree automata with an auxiliary memory containing ground terms. Let us assume two signatures: an input signature Σ and a memory signature Γ. We propose a definition of pushdown tree automata transitions in the clausal formalism, with Horn clauses of the following two forms.

$$q_1(x_1, s_1), \ldots, q_n(x_n, s_n) \Rightarrow q\big(f(x_1, \ldots, x_n), h(y_1, \ldots, y_m)\big) \qquad \text{(read)}$$

where $f \in \Sigma$, $h \in \Gamma$, $s_1, \ldots, s_n \in \mathcal{T}(\Gamma, \{y_1, \ldots, y_m\})$ (*read rules* in [Guessarian, 1983]), and

$$q_1(x, s) \Rightarrow q\big(x, h(y_1, \ldots, y_m)\big) \qquad \text{(pda-}\varepsilon\text{)}$$

where $h \in \Gamma$, $s \in \mathcal{T}(\Gamma, \{y_1, \ldots, y_m\})$ (ε-rules in [Guessarian, 1983]).

Let us call *pushdown tree automata* (PTA) a finite set of clauses of type (read) or (pda-ε). As usual, the (non-equational) predicate symbols are called states of the PTA. Note however that, at the difference of the previous sections, the states are binary for PTA, and not unary. Intuitively, the first argument of a state in a PTA is a subterm of the input term, and the second argument is the content of the memory.

Given a PTA $\mathcal{A}$ and a state q of $\mathcal{A}$, the *language* of $\mathcal{A}$ in state q, denoted $\mathcal{L}(\mathcal{A}, q)$, is the set of ground terms $t \in \mathcal{T}(\Sigma)$ such that $q(t, s)$ holds in the smallest Herbrand model of $\mathcal{A}$, for some $s \in \mathcal{T}(\Gamma)$. The *stack language* of $\mathcal{A}$ in state q, denoted $\mathcal{S}(\mathcal{A}, q)$, is the set of ground terms $s \in \mathcal{T}(\Gamma)$ such that $q(t, s)$ holds in the smallest Herbrand model of $\mathcal{A}$, for some $t \in \mathcal{T}(\Sigma)$.

Note that adding some variants of the rules (read) and (pda-ε) where the term $h(y_1, \ldots, y_m)$ in the right-hand-side is replaced by a single variable y does not increase the expressiveness of the model [Guessarian, 1983].

In [Guessarian, 1983], it is also shown that the memory signature Γ of PTA can be assumed to contain only nullary and unary function symbols, without loss of generality (*i.e.* the PTA with a stack memory have the same expressiveness as the PTA with a tree memory).

Visibly Pushdown Tree Automata

We have proposed with Hubert Comon-Lundh and Nicolas Perrin, a generalization [Comon-Lundh et al., 2007] of the visibility condition of [Alur and Madhusudan, 2004] for some tree automata transitions of the form of the above PTA. The idea is that the symbol in input (in a term in the case of [Comon-Lundh et al., 2007] or [Alur et al., 2006], and in a word in the case of [Alur and Madhusudan, 2004]) determines the kind of operation that the automata can perform on the stack. Therefore, we assume that the input signature Σ is partitioned in eight subsets, each subset corresponding to one category of operation.

$$\Sigma = \Sigma_{\mathsf{push}} \uplus \Sigma_{\mathsf{pop}_{11}} \uplus \Sigma_{\mathsf{pop}_{12}} \uplus \Sigma_{\mathsf{pop}_{21}} \uplus \Sigma_{\mathsf{pop}_{22}} \uplus \Sigma_{\mathsf{int}_0} \uplus \Sigma_{\mathsf{int}_1} \uplus \Sigma_{\mathsf{int}_2}$$

The *visibly pushdown tree automata* (VPTA) are finite sets of Horn clause of the types defined in Figure 4. For the sake of simplicity, we assume in this definition that all the symbols of Σ and Γ have either arity 0 or 2. This is not a real restriction, and the results of [Comon-Lundh et al., 2007] can be extended straightforwardly to the case of function symbols with other arity. We assume moreover that Γ contains a special constant symbol $\bot$, which is used to represent an empty memory. Note that the other constant symbols of Γ are not relevant since they cannot be pushed or popped. Every pop rule has a variant which read an empty memory.

In [Comon-Lundh et al., 2007], VPTA are called visibly tree automata with one memory following the terminology of [Comon and Cortier, 2005]. The VPTA strictly generalizes the VP Languages of [Alur and Madhusudan, 2004], and the addition of constraints on the memory contents.

The visibly tree automata of [Alur et al., 2006] use a word stack instead of a tree structured memory. The comparison with VPTA is not easy as they are alternating and compute top-down on infinite trees.

A formalism combining pushdown top-down tree automata of [Guessarian, 1983] with the concept of visibly pushdown languages has been proposed in [Chabin and Réty, 2007]. These automata recognize finite trees using a word stack. They have a decidable emptiness problem and the corresponding tree languages (Visibly Pushdown Tree Languages, VPTL) are closed under Boolean operations. Following remarks of one of the two authors of [Chabin and Réty, 2007], it appeared that VPTA and visibly pushdown tree languages are incomparable.

The class of languages of context-free tree grammars is not closed under intersection or complementation. The visibility condition of VPTA enables a Cartesian

Figure 4 Transitions of VPTA.

$$
\begin{array}{llcll}
q_1(x_1, y_1), & q_2(x_2, y_2) & \Rightarrow & q\big(f_2(x_1, x_2), h(y_1, y_2)\big) & (\mathsf{push}\ ) \\
q_1\big(x_1, h(y_{11}, y_{12})\big), & q_2(x_2, y_2) & \Rightarrow & q\big(f_3(x_1, x_2), y_{11}\big) & (\mathsf{pop}_{11}) \\
q_1(x_1, \bot), & q_2(x_2, y_2) & \Rightarrow & q\big(f_3(x_1, x_2), \bot\big) & \\
q_1(x_1, h(y_{11}, y_{12})), & q_2(x_2, y_2) & \Rightarrow & q\big(f_4(x_1, x_2), y_{12}\big) & (\mathsf{pop}_{12}) \\
q_1(x_1, \bot), & q_2(x_2, y_2) & \Rightarrow & q\big(f_4(x_1, x_2), \bot\big) & \\
q_1(x_1, y_1), & q_2\big(x_2, h(y_{21}, y_{22})\big) & \Rightarrow & q\big(f_5(x_1, x_2), y_{21}\big) & (\mathsf{pop}_{21}) \\
q_1(x_1, y_1), & q_2(x_2, \bot) & \Rightarrow & q\big(f_5(x_1, x_2), \bot\big) & \\
q_1(x_1, y_1), & q_2\big(x_2, h(y_{21}, y_{22})\big) & \Rightarrow & q\big(f_6(x_1, x_2), y_{22}\big) & (\mathsf{pop}_{22}) \\
q_1(x_1, y_1), & q_2(x_2, \bot) & \Rightarrow & q\big(f_6(x_1, x_2), \bot\big) & \\
 & & \Rightarrow & q(a, \bot) & (\mathsf{int}_0) \\
q_1(x_1, y_1), & q_2(x_2, y_2) & \Rightarrow & q\big(f_7(x_1, x_2), y_1\big) & (\mathsf{int}_1) \\
q_1(x_1, y_1), & q_2(x_2, y_2) & \Rightarrow & q\big(f_8(x_1, x_2), y_2\big) & (\mathsf{int}_2)
\end{array}
$$

where $q_1, q_2, q \in Q$, $x_1, x_2, y_1, y_2, \ldots$ are distinct variables of $\mathcal{X}$, $h \in \Gamma_2$, $a \in \Sigma_{\mathsf{int}_0}$, and every f_i is in the corresponding partition of Σ ($f_2 \in \Sigma_{\mathsf{push}}$, $f_3 \in \Sigma_{\mathsf{pop}_{11}}$, etc).

product construction for the intersection. More precisely, the memory signature, state set and final state set of the VPTA for intersection are the Cartesian products of the respective memory signatures, state sets and final state sets of the two VPTA in input, and the construction of product clauses is possible because two clauses with the same input function symbol (in Σ) in their head have the same shape.

Moreover, we propose in [Comon-Lundh et al., 2007] a determinization procedure for VPTA, with an exponential blowup in the number of states and symbols in the memory signature. The closure under complementation follows.

Theorem 17 [Comon-Lundh et al., 2007]

The class of VPTA tree languages is closed under union, intersection and complement.

Regarding the decision problems, in particular emptiness, we can observe that, given a VPTA $\mathcal{A}$ and one of its states q, the language $\mathcal{L}(\mathcal{A}, q)$ is empty iff the memory language $\mathcal{S}(\mathcal{A}, q)$ is empty. We show [Comon-Lundh et al., 2007] that every $\mathcal{S}(\mathcal{A}, q)$ is recognized by a two-way tree automaton, hence it is regular (see Section 2.6). The clauses of the two-way tree automaton are obtained simply by forgetting the first component of predicates in the clauses of the VPTA. Even better, the clauses obtained belong to the class $\mathcal{H}_3$ [Nielson et al., 2002], for which emptiness is decidable in cubic time. It follows that emptiness of VPTA is decidable in cubic time.

Theorem 18 [Comon-Lundh et al., 2008]

The emptiness problem is PTIME-complete for VPTA. The universality and inclusion problems are EXPTIME-complete for VPTA.

The results on universality and inclusion immediately follow from the closure results of Theorem 17 and the decidability of emptiness. The lower bounds follow from the corresponding properties for standard TA.

Visibly Pushdown Tree Automata with local constraints

With Hubert Comon-Lundh and Nicolas Perrin, we have proposed [Comon-Lundh et al., 2007; Comon-Lundh et al., 2008] to extend VPTA with local constraints testing the content of the memory. The idea is that, since each step of a bottom-up computation starts with two states and two memories (and ends with one state and one stack), it is possible to compare the contents of these two memories, with respect to an equivalence relation R on $\mathcal{T}(\Gamma)$. In Figure 5 we describe two types of clauses which are constrained versions of clauses int_1 and int_2, and induce two new categories for the symbols of Σ: int_1^R, int_2^R, in addition to the eight previous categories of Figure 4. We do not extend the clauses (push) and (pop) with constraints for some technical reasons explained in [Comon-Lundh et al., 2007].

Figure 5 Constrained variant of VPTA clauses.

$$q_1(x_1,y_1),\quad q_2(x_2,y_2),\quad R(y_1,y_2) \;\Rightarrow\; q\big(f_9(x_1,x_2),y_1\big) \qquad (\mathsf{int}_1^R)$$
$$q_1(x_1,y_1),\quad q_2(x_2,y_2),\quad R(y_1,y_2) \;\Rightarrow\; q\big(f_{10}(x_1,x_2),y_2\big) \qquad (\mathsf{int}_2^R)$$

The equivalence relation R is specified by some Horn clauses of the following form

$$R_1(x_{11},x_{12}), R_2(x_{21},x_{22}) \Rightarrow R_3\big(g(x_{11},x_{21}),h(x_{12},x_{22})\big) \qquad (\mathsf{bin})$$

where R_1,R_2, R_3 are auxiliary binary predicates (R is one of these predicates) and $g,h \in \Gamma$. We also assume in the type of clauses (bin) the cases where g or h have arity 0. A finite set $\mathcal{B}$ of Horn clauses of type (bin) always has a minimal Herbrand model (these clauses are all definite) and we can define as above the language $\mathcal{L}(\mathcal{B},R)$ of $\mathcal{B}$ in predicate R as the set of pairs of ground terms $s,t \in \mathcal{T}(\Sigma)$ such that $R(s,t)$ is true in the smallest Herbrand model of $\mathcal{B}$. A binary relation on ground terms is called *regular* if it is a union of languages of some finite set of clauses of type (bin).

We call *visibly pushdown tree automaton with R-constraints* (VPTA^R) a finite set containing Horn clauses of the types defined in Figures 4 and 5, as well as some clauses (bin) specifying the relation R.

The definition of [Comon-Lundh et al., 2007] actually also includes negative constraints, in clauses *e.g.* of the form $q_1(x_1,y_1), q_2(x_2,y_2), \neg R(y_1,y_2) \Rightarrow q\big(f_9(x_1,x_2),y_1\big)$. These are not Horn clauses, hence our semantics based on the smallest Herbrand

model is not adapted to this case (the definition of the languages in [Comon-Lundh et al., 2007] is based on more operational semantics). However, as shown in [Comon-Lundh et al., 2008], it is possible to get rid of negative constraints, providing that the cardinality of every equivalence classes modulo R is finite (this condition is satisfied by the relations of syntactic and structural equality described below). More precisely, we show that in this case, for every VPTA^R $\mathcal{A}$ with states set Q, extended with transitions with negative constraints as above, there exists a VPTA^R $\mathcal{A}'$ (without negative constraints) whose set of states is a superset of Q, and such that for all $q \in Q$, $\mathcal{S}(\mathcal{A}', q) = \mathcal{S}(\mathcal{A}, q)$.

Emptiness is in general undecidable for VPTA^R.

Theorem 19 [Comon-Lundh et al., 2008]

The membership is NP-complete and emptiness is undecidable for VPTA^R when R is a regular binary relation.

Emptiness becomes decidable when the memory languages are regular, and effectively recognized by a TA (remember that for VPTA^R, the emptiness of the language is equivalent to the emptiness of the memory language). In [Comon-Lundh et al., 2007], we propose the following criteria on R ensuring that $\mathcal{S}(\mathcal{A}, q)$ is regular for every state q of a VPTA^R $\mathcal{A}$: R is defined by a finite set of clauses (bin) over the state set $\{R_1, \ldots, R_n\}$ and such that $\forall i, j \ \exists k, l \ R_i(x, y) \wedge R_j(y, z) \nRightarrow R_k(x, y) \wedge R_l(x, z)$. It is shown by a technique of saturation with a calculus similar to the one of the above sections (resolution with selection and eager splitting), applied to the clauses obtained by forgetting the first argument of states (but not of the predicates R_i). This generalizes a former proof of [Comon and Cortier, 2005], see also the paragraph in Section 2.6 on the transformation of alternating and two-way TA into TA. Two classes of constraints satisfying this condition are the *syntactic equality* $=$ defined by

$$\begin{array}{rcll} & \Rightarrow & a = a & \forall a \in \Gamma_0 \\ x_{11} = x_{12}, x_{21} = x_{22} & \Rightarrow & g(x_{11}, x_{21}) = g(x_{12}, x_{22}) & \forall g \in \Gamma_2 \end{array}$$

and the *structural equality* $\equiv$ defined by

$$\begin{array}{rcll} & \Rightarrow & a \equiv b & \forall a, b \in \Gamma_0 \\ x_{11} \equiv x_{12}, x_{21} \equiv x_{22} & \Rightarrow & g(x_{11}, x_{21}) \equiv h(x_{12}, x_{22}) & \forall g, h \in \Gamma_2 \end{array}$$

Theorem 20 [Comon-Lundh et al., 2008]

Membership is NP-complete and emptiness is decidable in EXPTIME for $\mathsf{VPTA}^{=}$. Membership is decidable in PTIME and emptiness is decidable in 2-EXPTIME for $\mathsf{VPTA}^{\equiv}$.

Moreover, $\mathsf{VPTA}^{\equiv}$ (and its extension with negative constraints) is closed under intersection and complementation, but not $\mathsf{VPTA}^{=}$, for which universality is undecidable.

The regular tree languages and VPL are particular cases of VPTA languages. In some cases, the TAB of [Bogaert and Tison, 1992] can be simulated by $\mathsf{VPTA}^{=}$ pushing all the symbols in input up to (dis)equality tests. Some other examples of $\mathsf{VPTA}^{\equiv}$ extended with negative tests can be found in [Comon-Lundh et al., 2008], including balanced binary trees, powerlists and red-black (binary search) trees.

Finally, in [Comon-Lundh et al., 2008], we also study the addition to VPTA^{R} of the TAB equality and disequality constraints à la [Bogaert and Tison, 1992] (testing the term in input, *i.e.* the variables in the first components of the states in clauses). We show the Boolean closure and the decidability of emptiness for the model obtained. Table 2 summarizes the results on VPTA presented in this section.

	$\in$	$\emptyset$	$\cap$, $\neg$
VPTA	PTIME	PTIME	yes
$\mathsf{VPTA}^{=}$	NP-complete	EXPTIME	no
$\mathsf{VPTA}^{\equiv}$	PTIME	2-EXPTIME	yes
VPTA^{R}	NP-complete	undec.	yes

Table 2: Summary of the results on VPTA.

2.6 Related Models

The above results may be alternatively viewed as new decidable classes of first-order formula. Other related decidable fragment of first-order logic include the extended Skolem class and the $\mathcal{E}^+$ class [Fermüller et al., 2001]. We present below some other decidable classes of clausal tree automata.

Two-Way and Alternating Tree Automata

The proof of Theorem 12 is based on an invariant on the type of clauses produced by the saturation, which contains only a finite (exponential) number of clauses. Let us present two particular subcases of this invariant, which correspond to the transitions of two classical tree automata models.

The application, by paramodulation, of an equational clause of type (eq) into an automatic clause (aut) can return a clause of the following form

$$q\big(f(x_1,\ldots,x_n)\big) \Rightarrow q_i(x_i) \qquad \text{(bidi)}$$

Such clauses, called two-way clauses in [Frühwirth et al., 1991], push clauses in [Verma, 2003] or *selecting theories* in [Truderung, 2005] correspond to transitions of two-way tree automata: the automatic clauses (aut) correspond to bottom-up transitions, whereas clauses as above correspond to top-down transitions.

The following kind of clause is also a particular case of the invariant type of clauses for the proof of Theorem 12

$$q_1(x), \ldots, q_n(x) \Rightarrow q(x) \qquad \text{(alt)}$$

Such a clause expresses that the state q will capture the intersection of the languages of $q_1, \ldots, q_n$. It is called an *alternating* clause, because it permits us to describe naturally alternating tree automata, an exponentially succinct presentation of tree automata.

Alternating and two-way tree automata are made of clauses of types (aut), (bidi) and (alt). They are equivalent in expressiveness to regular tree languages, and there is an exponential transformation procedure from these automata to standard tree automata, see [Frühwirth et al., 1991; Charatonik and Podelski, 2002], and also [Comon et al., 2007] (Chapter 7).

The transformation can be achieved by saturation techniques (by resolution) [Goubault-Larrecq, 2006]. In our framework, as noticed above, the clauses (bidi) and (alt) are particular cases of the invariant type of clauses generated by the basic ordered paramodulation with selection and ε-splitting, starting from a TAE. Hence, when started with a finite set of automatic clauses of types (aut), (bidi) and (alt) saturation by this calculus will terminate with a set $\mathcal{C}$ of clauses of the invariant type. Moreover, it can be shown that, thanks to the chosen ordered and selection strategies, for every ground term $t \in \mathcal{T}(\Sigma)$ and state q occurring in $\mathcal{C}$, a paramodulation step between a clause of $\mathcal{C}$ and the goal clause $q(t) \Rightarrow$ can only involve an automatic clause (the other clauses of the invariant type are excluded by the ordering or selection condition). Hence, the tree language defined by $\mathcal{C}$ is regular, and it is recognized by a tree automaton made of the subset of automatic clauses (aut) of $\mathcal{C}$.

The class $\mathcal{H}_1$

A class of finite Horn clause sets called $\mathcal{H}_1$ has been proposed in [Nielson et al., 2002], for which satisfiability is EXPTIME-complete, and such that $\mathcal{H}_1$ clause sets can be transformed to standard TA in exponential time. The subclasses $\mathcal{H}_2$ and $\mathcal{H}_3$ have respectively polynomial time and cubic satisfiability. These results are applied to the control flow analysis of the Spi-calculus.

In the version of [Goubault-Larrecq, 2005], $\mathcal{H}_1$ Horn clauses have a head whose argument is at most of height one and linear (without duplicated variables), or are purely negative (goals), and the conversion to TA is obtained by saturation techniques (ordered resolution with selection and ε-splitting). $\mathcal{H}_1$ becomes undecidable when allowing variable duplication in the heads. The above model TAD allows this under the previously mentioned restrictions.

Tree Automata with Term Constraints

The *bottom-up tree automata with term constraints* (TCA) of [Reußand Seidl, 2010] are defined as finite set of Horn clauses of the form

$$q_1(x_1), \ldots, q_n(x_n), x_{i_1} = s_1, \ldots, x_{i_k} = s_k, x_{j_1} \neq t_1, \ldots, x_{j_l} \neq t_l \Rightarrow q\big(f(x_1, \ldots, x_n)\big) \tag{tca}$$

where $1 \leq i_1, \ldots, i_k, j_1, \ldots, j_l \leq n$ and $s_1, \ldots, s_k, t_1, \ldots, t_l \in \mathcal{T}(\Sigma, \{x_1, \ldots, x_n\})$.

The TCA strictly generalizes the TAB of [Bogaert and Tison, 1992]. Without the condition that the variables of the terms s_i in the equalities are restricted to $x_1, \ldots, x_n$, the automata would be able to simulate the undecidable class TAC. The class of TCA languages is closed under Boolean operation, and has an exponential determinization procedure. Note that the TCA are interesting for the characterization of languages of normal form *wrt* a TRS, and therefore the results of [Reußand Seidl, 2010] could be useful in particular in the context of inductive theorem proving, see Section 1.5.

It is shown in [Reußand Seidl, 2010] that the the equality constraints in clauses (tca) can be eliminated, and that emptiness is decidable for TCA with only disequality constraints. The decision algorithm tries to construct witnesses for the languages of the different states of the automaton, and the number of necessary iterations is bounded using a combinatorial argument.

Perspectives: TCA and saturation

It would be interesting to know whether some classical saturation techniques as the ones of the above section could be used to decide emptiness of TCA, and whether it is possible to combine the TCA clauses with equalities clauses (eq) or clauses with equality tests (deep) in a decidable model.

Horn Clauses with Rigid Variables

Rigid clauses [Andrews, 1981] contain distinguished free variables called *rigid*, and denoted by uppercase letters. A set $\mathcal{C}$ of clauses with rigid free variables $Y_1, \ldots, Y_m$ and other free variables $x_1, \ldots, x_n$ (called *flexible*) is satisfiable if there exists a Σ-algebra $\mathfrak{A}$ such that for all valuation $\sigma : \{X_1, \ldots, X_m\} \to \mathfrak{A}$, there exists a first order model $\mathcal{M}$ with domain $\mathfrak{A}$ such that $\mathcal{M}, \sigma \models \forall x_1, \ldots, x_n \mathcal{C}$. It is equivalent to say that for all valuation $\sigma : \{X_1, \ldots, X_m\} \to \mathcal{T}(\Sigma)$, there exists an Herbrand model $\mathcal{H}$ such that $\mathcal{H} \models \forall x_1, \ldots, x_n \; \sigma(\mathcal{C})$.

A translation of clauses with rigid variables into first order clauses (without rigid variables), preserving satisfiability, is proposed in [Affeldt and Comon-Lundh, 2009], and used below in Section 3.2. The idea is to add accumulating parameters to the predicates in order to store the valuations of the rigid variables. Each rigid variable is evaluated once for all, hence the parameters can be set only once, as indicated by a flag in the predicate name, with a clause of the form

$$q_{\overline{u}}(y_i, y_1, \ldots, y_m) \Rightarrow q_{\overline{u}'}(y_i, y_1, \ldots, y_m) \tag{write}$$

where the variables $y_1, \ldots, y_m$ are the accumulating parameters, and are pairwise distinct, and $\overline{u}, \overline{u}' \in \{0,1\}^m$ are tuples of flags such that $u_i = 0$, and $u'_i = 1$ and $u'_j = u_j$ for all $j \neq i$. Note the similarity between the relation on the tuples of flags and the ordering condition on the predicates of clause (deep), in Section 2.3.

Otherwise, the parameters are propagated in the clauses, like for instance in the following extension of automatic clause, with m parameters

$$q_{\overline{u}}(x_1, y_1, \ldots, y_m), \ldots, q_{\overline{u}}(x_n, y_1, \ldots, y_m) \Rightarrow q_{\overline{u}}\big(f(x_1, \ldots, x_n), y_1, \ldots, y_m\big) \qquad \text{(rig)}$$

where the variables $x_1, \ldots, x_n$, $y_1, \ldots, y_m$ are pairwise distinct.

The clauses studied in [Affeldt and Comon-Lundh, 2009] are actually more general than this, with *e.g.* arbitrary terms s and t instead of the last occurrences of y_i in the body and the head of (write). However, although the name of tree automata is not used in [Affeldt and Comon-Lundh, 2009], such clauses can alternatively be seen as transitions of tree automata extended with m auxiliary registers storing terms of $\mathcal{T}(\Sigma)$. It is shown in [Affeldt and Comon-Lundh, 2009] that binary resolution with an appropriate ordered strategy terminates on clauses of a certain type, including the above (write) and (rig). Section 3.2 discusses the relation between this result and classes of tree automata with global equality constraints.

2.7 Application to the verification of communicating processes

One of our motivations for the introduction of TADE was to represent concurrent processes exchanging messages asynchronously over insecure communication channels. In some models like the applied π-calculus [Abadi and Fournet, 2001], the messages are not atoms but ground terms, and the processes construct the messages they send from the messages they can read in the network, by application of some operators whose semantics are defined by equational axioms. Some examples of operators and axioms are given in Example 12. The insecure communication channels can be represented in this setting by their ability to alter the messages exchanged, also by application of operators (to existing messages). Therefore, the number and the size of the messages that can be found at some time in the communication channels is unbounded, and it is relevant to use tree automata for representing these sets of messages.

Tree automata techniques have actually been quite a popular approach for the static analysis of such systems, in particular for deciding reachability properties, see *e.g.* [Comon and Cortier, 2005; Genet and Klay, 2000; Goubault-Larrecq, 2005; Verma and Goubault-Larrecq, 2007a; Seidl and Verma, 2008, 2009] But this kind of analyses has limitations due to approximations with regular sets. Such approximations may conduct to false alarms, as discussed *e.g.* in [Amadio and Charatonik, 2002] or [Affeldt and Comon-Lundh, 2009]. The goal for the introduction of the TADE in this context was to avoid some sources of imprecision, with an increased expressiveness (over TA), in order to handle the multiple occurrences of variables in the body of messages sent with clauses (deep), and with a better control of the interleaving of messages sent by processes, using several state symbols. Moreover, TADE permit us to represent the equational axioms directly in a uniform model, as equational clauses

(eq) of TAE, and to enable conditional tests for the composition of messages, with constrained clauses (deep) of TAD. We will give below some more descriptions of the principles of the model with TADE (without too much details though), and discuss possible further directions.

The system to verify is a finite set of concurrent processes, where a *process* is a finite sequence of *instructions* of the form $\mathsf{recv}(x).\mathsf{if}\, e\, \mathsf{then}\, \mathsf{send}(s)$ where x is a variable, $s \in \mathcal{T}(\Sigma, \{x\})$ and e is a set (possibly empty) of equations between terms of $\mathcal{T}(\Sigma, \{x\})$. The sub-instruction recv acts as a binder for the variable x. We also assume a program point before each instruction, and a distinguished program point at the end of the process. Moreover, a finite set $\mathcal{E}$ of sublinear and collapsing equations (see Example 12) is also assumed given, and all the terms are considered modulo this theory. The signature Σ contains two kind of symbols: the public symbols, representing functions publicly known, and private symbols, representing secret constants or functions. In the signature of Example 12, pair, fst, snd, enc, dec, aenc, adec, pub are public and inv is private. Given $N \subseteq \mathcal{T}(\Sigma)$, we define $\mathit{closure}_{\mathcal{E}}(N)$ as the smallest (*wrt* inclusion) subset of $\mathcal{T}(\Sigma)$ containing N, closed under application of equations of $\mathcal{E}$, and such that for all public symbol $f \in \Sigma_n$, for all $t_1, \ldots, t_n \in \mathit{closure}_{\mathcal{E}}(N)$, $f(t_1, \ldots, t_n) \in \mathit{closure}_{\mathcal{E}}(N)$.

A *configuration* of the system is a pair (st, N) where st is a function associating to each process its current program point, and N is a (possibly infinite) set of ground terms representing the possible content of the communication channel (we assume only one channel for simplicity but could model several). The operational semantics for the system is the following (we purposely omit some details): every execution step changes the running configuration (st, N) into (st', N'), if for some process p, the program point $st(p)$ is not the final program point of p, the instruction after this program point is $\mathsf{recv}(x).\mathsf{if}\, e\, \mathsf{then}\, \mathsf{send}(s)$, $st'(p)$ is the next program point of p, $st'(p') = st(p')$ for all process $p' \neq p$, there exists a ground term $t \in \mathit{closure}_{\mathcal{E}}(N)$ and the substitution $\theta = \{x \mapsto t\}$ is such that $\theta(\ell)$ and $\theta(r)$ are equal modulo $\mathcal{E}$ for all $\ell = r$ in e, and $N' = N \cup \{\theta(s)\}$. We assume an initial configuration (st_0, N_0) where $st_0(p)$ is the first program point of p for every process p and N_0 is an arbitrary regular tree language over Σ.

Example 13 *Let $\mathcal{E}$ be the set of sublinear and collapsing equations of Example 12. The following instruction* $0.\mathsf{recv}(x).\mathsf{if}\, \mathsf{fst}(t) = \mathsf{b},\, \mathsf{then}\, \mathsf{send}(\mathsf{enc}(\mathsf{m}, \mathsf{snd}(t))).1$ *with* $t = \mathsf{adec}\big(\mathsf{adec}(\mathsf{snd}(x), \mathsf{pub}(\mathsf{fst}(x))), \mathsf{inv}(\mathsf{pub}(\mathsf{b}))\big)$*, expresses that the process receives a message x, then it verifies, with the equations in the conditional, that x has the form* $\mathsf{pair}\big(\mathsf{a}, \mathsf{aenc}(\mathsf{aenc}(\mathsf{pair}(\mathsf{b}, \mathsf{key}), \mathsf{pub}(\mathsf{b})), \mathsf{inv}(\mathsf{pub}(\mathsf{a})))\big)$*. The symbols* a, b, key *are constant and private function symbols. The second component of the message x is a double encryption: once with the public key* pub(b)*, and once with the private key of* inv(pub(a)) *(the latter corresponds to a signature by* a*). According to the equations of $\mathcal{E}$, it means that t is equal (modulo $\mathcal{E}$) to* pair(b, key) *and hence* snd(t) = key*. This value* key *is then use to encrypt a confidential message* m *sent by the process.* ◇

If N is regular, then $\mathit{closure}_{\mathcal{E}}(N)$ is a regular tree language modulo $\mathcal{E}$ (see *e.g.*

[Amadio and Lugiez, 2000]). It is actually possible to construct a TADE $\mathcal{A}$ whose language is the set of terms t in N for a reachable configuration (st, N). This TADE contains the equations of $\mathcal{E}$, as clauses of type (eq). It contains also, for each instruction of the form $i.\mathsf{recv}(x).\mathsf{if}\, u_1 = v_1, \ldots, u_k = v_k\, \mathsf{then}\, \mathsf{send}(s).i'$, in a process p, the clauses of type (deep) of the form

$$q_{\overline{m},i,\overline{n}}(x), u_1 = v_1, \ldots, u_k = v_k, y = s \Rightarrow q_{\overline{m},i',\overline{n}}(y).$$

The indexes i and i' are program points of p and $\overline{m}$, $\overline{n}$ are arbitrary sequences of program points of the other processes. Intuitively, the state $q_{\overline{m},i,\overline{n}}$ represents the possible contents of the communication channel when the processes are at the respective program points $\overline{m}, i, \overline{n}$. It is immediate to find an ordering on state predicates such that $q_{\overline{m},i',\overline{n}} > q_{\overline{m},i,\overline{n}}$, ensuring the conditions of type (deep).

Example 14 *Let us come back to Example 13, assuming one process p_1 with program points $\{0, 1, 2\}$ and one other process p_2 containing only the instruction in this example (with program points 0 and 1). The clauses of type (deep) associated with this systems are (with $i = 0, 1, 2$ and $t = \mathsf{adec}\big(\mathsf{adec}(\mathsf{snd}(x), \mathsf{pub}(\mathsf{fst}(x))), \mathsf{inv}(\mathsf{pub}(\mathsf{b}))\big)$)*

$$q_{i,0}(x), \mathsf{fst}(t) = \mathsf{b}, y = \mathsf{enc}(\mathsf{m}, \mathsf{snd}(t)) \Rightarrow q_{i,1}(y)$$

◇

The TADE $\mathcal{A}$ also contains some clauses of type (aut′) and (deep) specifying the closure $closure_{\mathcal{E}}(N)$, like *e.g.* $q_{\overline{m}}(x_1), q_{\overline{m}}(x_2) \Rightarrow q_{\overline{m}}(\mathsf{f}(x_1, x_2))$ where $f \in \Sigma$ is a public symbol, or $q_{\overline{m}}(x_1), q_{\overline{n}}(x_2) \Rightarrow q_{\overline{n}}(\mathsf{f}(x_1, x_2))$ with $\overline{n} > \overline{m}$. Finally, it also contains some clauses of type (aut′) for the definition of N_0, part of the initial configuration.

In [Jacquemard et al., 2008a], we present with Michael Rusinowitch and Laurent Vigneron two authentication protocols modeled as TADE following the above principles, including a recursive authentication protocol [Bull and Otway, 1997] which ensures the distribution of certified session keys to a group of clients by a server which process recursively an unbounded list of requests. The model is then used to verify some reachability properties of the protocol using the termination results presented in Section 2.4. This serves either at finding an attack, when the empty clause is derived, or certify the protocol otherwise.

Perspectives: extensions of the TADE model

The TADE clauses of Section 2.4 can be extended in several ways in order to improve this model. For instance, an extension with disequality constraints (in addition to the current equality constraints of clauses (deep) would permit us to model if-then-else conditional, and not only if-then as above (see the above discussion in Section 2.6).

It would also be interesting to extend the above saturation results (in particular for classes modulo monadic or collapsing theories) to term algebra modulo associativity and commutativity (AC), using AC-paramodulation techniques [Nieuwenhuis and Rubio,

2001]. This combination (AC + sublinear–collapsing theories) would permit ones *e.g.* to axiomatizing operators like the exclusive-or.

Another more challenging perspective is to extend TADE in order to handle processes with local and global memories taking their values in infinite domains and which can be written only once. A limitation of the above model of processes is indeed that the term s and equations e of an instruction $\mathsf{recv}(x).\mathsf{if}\, e\, \mathsf{then}\, \mathsf{send}(s)$ can only use the variable x (the message that was immediately read). The goal of the extension would be to enable to reuse in s and e some other previously bound variables. In the semantics, a configuration is in this case a triple (st, σ, N) where st and N are as above, and σ is a substitution of the bound variables of the processes which are assumed pairwise distinct (we studied such a model with Stephanie Delaune in [Delaune and Jacquemard, 2004]). A natural model for such processes would be an extension of TADE-like clauses with accumulating parameters (*rigid variables*) like in the clauses presented in Section 2.6 (see also the discussion in Section 3.2). These accumulating parameters are also necessary in order to obtain an exact model for the representation of information that can be exchanged by the processes, as they prevent to make several instantiation of the same bound variable x occurring in an instruction $\mathsf{send}(x)$, which is a cause of false positive, see [Affeldt and Comon-Lundh, 2009].

Our initial goal for the studies presented above was actually to propose a clausal tree automaton model combining local equality and disequality constraints like RA with rigid variables, in order to obtain an exact abstract model of communicating processes as above. Then termination results for the saturation of such clauses *wrt* to a first order calculus could enable the proof of safety properties, or temporal properties, by intersection with a TA language (hence using classical regular model checking techniques). Unfortunately, only the first step of this project (study and implementation of TADE) could be realized.

3 Tree Automata with Global Constraints

The tree automata presented in Section 1 test some constraints on ranked terms at runtime, during the application of transition rules. Another more recent trend in tree automata theory consists in testing a *global constraint* only once, at the end of the computation of the automaton, where some equalities and disequalities are checked between some subterms at positions defined by the states reached by the automaton during the computation. We present in this section some results in this approach. We use a general tree automata model with global constraints, TAGC, introduced with Luis Barguñó, Carlos Creus, Guillem Godoy, and Camille Vacher in [Barguñó et al., 2010], and present former classes, such as TAGED [Filiot et al., 2008], as particular cases of TAGC.

Let us consider some constraints interpreted on runs of TA. An atomic *equality constraint* (resp. atomic *disequality constraint*) over a state set Q is a pair of states denoted $q \approx q'$ (respectively $q \not\approx q'$) with $q, q' \in Q$. It is satisfied by a run r of a

TA on a term $t \in \mathcal{T}(\Sigma)$, denoted by $r \models q \approx q'$ (respectively $r \models q \not\approx q'$) if for all different positions $p, p' \in \mathcal{P}os(t)$ such that $r(p) = q$ and $r(p') = q'$, $t|_p = t|_{p'}$ holds (respectively $t|_p \neq t|_{p'}$ holds). The satisfiability is extended to Boolean combination of atomic constraints as expected. Note that the semantics of $\neg(q \approx q')$ and $q \not\approx q'$ differ, as well as the semantics of $\neg(q \not\approx q')$ and $q \approx q'$.

We denote by $\approx$ the type of equality constraints, $\not\approx$ the type of disequality constraints. These types are further partitioned into the type $\approx_{\mathsf{irr}}$ of *irreflexive equality* constraints: atomic constraints of the form $q \approx q'$ with $q \neq q'$, the type $\not\approx_{\mathsf{irr}}$ of *irreflexive disequality* constraints: atomic constraints of the form $q \not\approx q'$ with $q \neq q'$, the type $\approx_{\mathsf{ref}}$ of *reflexive equality* constraints: atomic constraints of the form $q \approx q$, and the type $\not\approx_{\mathsf{ref}}$ of *reflexive disequality* constraints: atomic constraints of the form $q \not\approx q$.

A *tree automaton with global constraints* $\mathsf{TAGC}[\tau_1, \ldots, \tau_k]$ over a signature Σ is a tuple $\mathcal{A} = \langle \Sigma, Q, F, \Delta, C \rangle$ where $\langle \Sigma, Q, F, \Delta \rangle$ is a TA, denoted $ta(\mathcal{A})$, and C is a Boolean combination of atomic constraints of type in $\tau_1, \ldots, \tau_k$.

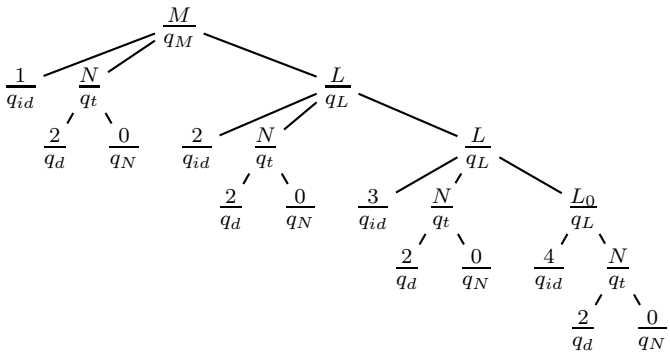
A *run* r of the $\mathsf{TAGC}[\tau_1, \ldots, \tau_k]$ $\mathcal{A} = \langle \Sigma, Q, F, \Delta, C \rangle$ is a run of $ta(\mathcal{A})$ such that furthermore $r \models C$. The run r is called *successful* (or *accepting*) if the state symbol $r(\varepsilon)$ at its root is in F. The language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of terms $t \in \mathcal{T}(\Sigma)$ on which there exists a successful run of $\mathcal{A}$. For every state $q \in Q$, we denote $\mathcal{L}(\mathcal{A}, q)$ the language of $\mathcal{A}$ in state q, which is the set of terms $t \in \mathcal{T}(\Sigma)$ on which there exists a run r of $\mathcal{A}$ with $r(\varepsilon) = q$. Hence $\mathcal{L}(\mathcal{A}) = \bigcup_{q \in F} \mathcal{L}(\mathcal{A}, q)$.

Example 15 *The following* $\mathsf{TAGC}[\approx, \not\approx]$ $\mathcal{A} = \langle Q, \Sigma, F, \Delta, \emptyset, C \rangle$ *accepts (in state* q_M*) lists of dishes called menus, where every dish is associated with one identifier (state* q_{id}*) and the time needed to cook it (state* q_t*). We have other states accepting digits (*q_d*), numbers (*q_N*) and lists of dishes (*q_L*). It is defined as follows:* $\Sigma = \{0, \ldots, 9 : 0, N, L_0 : 2, L, M : 3\}$, $Q = \{q_d, q_N, q_{id}, q_t, q_L, q_M\}$, $F = \{q_M\}$, *and* $\Delta = \{i \to q_d \mid q_N \mid q_{id} \mid q_t : 0 \leq i \leq 9\} \cup \{N(q_d, q_N) \to q_N \mid q_{id} \mid q_t, L_0(q_{id}, q_t) \to q_L, L(q_{id}, q_t, q_L) \to q_L, M(q_{id}, q_t, q_L) \to q_M\}$. *The constraint* C *ensures that all the identifiers of the dishes in a menu are pairwise distinct (i.e. that* q_{id} *is a key) and that the time to cook is the same for all dish:* $C = q_{id} \not\approx q_{id} \wedge q_t \approx q_t$. *A term in* $\mathcal{L}(\mathcal{A})$ *together with an associated successful run are depicted in Figure 6.* ◇

3.1 TAGED

The TAGED model (*tree automata with global equality and disequality constraints*) has been introduced in [Filiot et al., 2007] as a tool for deciding satisfiability of a spatial logic. It is equivalent to the subclass of $\mathsf{TAGC}[\approx, \not\approx_{\mathsf{irr}}]$ whose constraints are conjunctions of atomic constraints (from now on, we shall call such a subclass *positive* $\mathsf{TAGC}[\approx, \not\approx_{\mathsf{irr}}]$). The class of TAGED languages is closed under union and intersection, but not under complement, and TAGED cannot be determinized [Filiot et al., 2008]. Membership is NP-complete for TAGED [Filiot et al., 2008], and the emptiness problem was first shown decidable for the following subclasses

Figure 6 Term and successful run (Example 15).



- TAGED with only equalities (*i.e.* positive TAGC[$\approx$]), for which emptiness is EXPTIME-complete [Filiot et al., 2008], see also the paragraph on RTA below,
- TAGED with only disequalities (*i.e.* positive TAGC[$\not\approx_{\mathsf{irr}}$]), for which emptiness is decidable in NEXPTIME [Filiot et al., 2008], see also the paragraph on DAG automata below,
- a subclass of TAGED which allow only a bounded number (by some constant independent of the tree) of (dis)equality tests on the run [Filiot et al., 2007],
- an extension of the above subclass to TAGED restricting to a bounded number the tests of disequality only, [Filiot et al., 2008].

Universality and inclusion are undecidable for TAGED [Filiot et al., 2008], and finiteness is decidable in EXPTIME [Filiot et al., 2008] for the first above subclass (TAGED with only equalities). The question of the decidability of emptiness for the whole class TAGED has been open for 3 years and we answered it by the positive (see Corollary 26 below).

3.2 Rigid Tree Automata

The *rigid tree automata* (RTA) form a further restricted subclass of TAGED, consisting in positive TAGC[$\approx_{\mathsf{ref}}$]. We introduced them with Francis Klay and Camille Vacher in [Jacquemard et al., 2009], in the context of reachability analysis for communicating processes (see Section 2.7). Note that RTA are sufficiently expressive to characterize the sets of ranked terms embedding a ground instance of a given (non-linear) term t as a subterm. It was already observed in [Filiot et al., 2008] that RTA already have the expressiveness of TAGC[$\approx$], but at the price of an exponential blowup.

We show in [Jacquemard et al., 2009] several results on the expressiveness of RTA, some of them are summarized below (DRTA is the class of deterministic RTA).

Theorem 21 [Jacquemard et al., 2009]

– The class of RTA languages is closed under union and intersection, with an exponential lower bound for the intersection.
– The class of RTA languages is not closed under complement.
– TA $\subsetneq$ DRTA $\subsetneq$ RTA.

The languages of RTA and $\mathsf{TA}_{=}$ (local constraints) are incomparable.
Regarding decidability, we have the following results.

Theorem 22 [Jacquemard et al., 2009]

– Membership is NP-complete for RTA.
– Emptiness is decidable in linear time for RTA.
– Universality and inclusion are undecidable for RTA.
– Finiteness is decidable in PTIME for RTA.

For the emptiness problem, one can observe that the emptiness of the language of a given RTA is equivalent to the emptiness of the language of its underlying TA.

A comparison of these results with some related models is presented in Table 3. Following Theorems 3 and 22 (undecidability of universality), regularity is undecidable for RTA, and therefore for the superclasses of TAGED and TAGC[$\approx$].

Theorem 23 [Barguñó et al., 2010]

Regularity is undecidable for RTA, TAGED, TAGC[$\approx$].

Our main contribution on RTA languages is the study of their rewrite closure, in [Jacquemard et al., 2009, 2011a]. These results are presented in Part II.1.4.

Perspective: Clausal Representation of RTA

We show in [Jacquemard et al., 2011a], with Francis Klay and Camille Vacher, that the languages of RTA can be defined as models of automatic clauses with rigid variables (see Section 2.6). The idea is to associate one rigid variable X_q to each rigid state q in the clausal representation of tree automata transition (see Section 2.1). More precisely, let $\mathcal{A} = \langle \Sigma, Q, F, \Delta, C \rangle$ be a RTA and let $R = \{q \in Q \mid q \approx q \in C\}$. We associate to $\mathcal{A}$ the set $\mathcal{C}_\mathcal{A}$ of rigid Horn clauses of the form $q_1(\alpha_1), \ldots, q_n(\alpha_n) \Rightarrow q\big(f(\alpha_1, \ldots, \alpha_n)\big)$ such that $f(q_1, \ldots, q_n) \to q \in \Delta$ and for all $i \leq n$, $\alpha_i = X_{q_i}$ if $q_i \in R$ and α_i is a flexible variable y_i otherwise. Then, we have

$$\bigcup_{\sigma : \{X_q \mid q \in R\} \to \mathcal{T}(\Sigma)} L\big(\sigma(\mathcal{C}_\mathcal{A}), q\big) = \mathcal{L}(\mathcal{A}, q).$$

Following [Affeldt and Comon-Lundh, 2009] (see Section 2.6) the translation of the above rigid automatic clause into a Horn clause without rigid variables returns

$$q_1(\alpha'_1, \overline{y}), \ldots, q_n(\alpha'_n, \overline{y}) \Rightarrow q\big(f(\alpha'_1, \ldots, \alpha'_n), \overline{y}\big) \qquad (\mathsf{rta})$$

where $\overline{y} = (y_q)_{q \in R}$ is a sequence of $|R|$ flexible variables, one variable y_q for each rigid state $q \in R$ (hence one for each rigid variable X_q). Every variable α'_i, $i \leq n$, is either y_{q_i} if α_i is the rigid variable X_{q_i} (*i.e.* if $q_i \in R$) and $\alpha'_i = \alpha_i = y_i$ otherwise.

A natural question is whether it is possible to saturate sets of clauses representing RTA, with *e.g.* the standard first order theorem proving techniques presented in Section 2. Note however that the termination result of [Affeldt and Comon-Lundh, 2009] (Section 2.6) does not enable this in full generality, because every clause (rig) involve only a single predicate (state) symbol, which is a strong restriction from an automata theoretic point of view. The resolution strategy of [Affeldt and Comon-Lundh, 2009] does not terminate on clauses (rig) with more than one state symbol and finding a terminating resolution strategy for this case seems not obvious. Some progress in this direction would enable the application of first order theorem proving techniques in order to decide problems for RTA, or extensions of RTA with *e.g.* equational tests or language modulo equational theories, with clauses generalizing (deep) and (eq) of Sections 2.3, 2.2 (see also Section II.1.4 regarding this latter point).

3.3 DAG Automata

A popular compact representation of ranked terms uses *directed acyclic graphs* (DAGs) with one unique root and where multiple occurrence of the same subtree are shared. This enables an exponential compression of terms in size (for instance in the case of balanced binary trees).

More precisely, the DAG associated to a term $t \in \mathcal{T}(\Sigma)$, denoted $dag(t)$, has for domain (set of nodes, denoted $dom(dag(t))$) the set of subterms of t, every node v of the form $f(s_1, \ldots, s_n)$ $(n \geq 0)$ has a label $f \in \Sigma$ (denoted $lab(v)$) and n successor nodes $succ_1(v) = s_1, \ldots, succ_n(v) = s_n$. Note that the edges outgoing of a node $f(s_1, \ldots, s_n)$ are ordered (from 1 to n). This kind of ordered DAGs are sometimes called *t-dags* in the literature, and we shall use this term from now on. With the above definition, the unique root of the t-dag $dag(t)$ is t itself. Let us denote $\mathcal{D}(\Sigma)$ the set $\{dag(t) \mid t \in \mathcal{T}(\Sigma)\}$. We extend the definition of dag from $\mathcal{T}(\Sigma)$ to the subsets of $\mathcal{T}(\Sigma)$ by $dag(L) = \{dag(t) \mid t \in L\}$.

In [Charatonik, 1999], Witold Charatonik proposes to run TA directly on the t-dags of $\mathcal{D}(\Sigma)$. The idea is to evaluate compressed terms with a TA without decompressing. In this case, the TA is called a *DAG automaton* (DA). More precisely, a *run* r of a TA $\mathcal{A}$ on a t-dag $d \in \mathcal{D}(\Sigma)$ is a function from $dom(d)$ into $Q_\mathcal{A}$ such that for each $v \in dom(d)$ with $lab(v) \in \Sigma_n$ $(n \geq 0)$, $lab(v)\big(r(succ_1(v)), \ldots, r(succ_n(v))\big) \to r(v)$ is a transition rule of $\Delta_\mathcal{A}$. The languages $\mathcal{L}_d(\mathcal{A}, q)$ and $\mathcal{L}_d(\mathcal{A})$ are the sets of t-dags on which there exists a run of $\mathcal{A}$ rooted by q, respectively by a final state of $\mathcal{A}$.

We can immediately associate to every run r of $\mathcal{A}$ on d a run r' of $\mathcal{A}$ on $dag^{-1}(d)$ with the same state at the root. In particular, for all t-dag $d \in \mathcal{L}_d(\mathcal{A})$, it holds that

$dag^{-1}(d) \in \mathcal{L}(\mathcal{A})$. However, the converse does not hold unless $\mathcal{A}$ is deterministic. For every TA $\mathcal{A}$, $dag\big(\mathcal{L}(\mathcal{A})\big)$ is the t-dag language of a DA (the deterministic version of $\mathcal{A}$), but there exists some (non-deterministic) DA such that $dag^{-1}\big(\mathcal{L}_d(\mathcal{A})\big)$ is not a regular tree language.

The class of DA languages is closed under union and intersection but not under complementation [Anantharaman et al., 2005] (the set of balanced binary trees is not DA-recognizable but its complement is DA-recognizable).

The problem of membership is NP-complete for DA: a non-deterministic decision procedure is presented in [Charatonik, 1999] and NP-hardness is proved in [Anantharaman et al., 2005] by reduction of Boolean satisfiability. An uncompressed version of this problem (given a TA $\mathcal{A}$ and a tdag d, decide whether $dag^{-1}(d) \in \mathcal{L}(\mathcal{A})$ was proved PTIME-complete in [Lohrey and Maneth, 2006].

The emptiness problem is also NP-complete for DA, as shown in [Charatonik, 1999] with a rather involved pumping argument, close to negative set constraint solving techniques [Charatonik and Pacholski, 1994, 2010]. Roughly, it is shown that if $\mathcal{L}_d(\mathcal{A})$ is not empty, then $\mathcal{A}$ recognizes a tdag of size at most a polynomial in the number of states of $\mathcal{A}$. Universality and inclusion are undecidable for DA [Anantharaman et al., 2005].

There is some kind of duality between the DA and the above classes of positive TAGC[$\approx$] (TAGED without disequalities) and RTA: the DA force in their runs a unique state for the multiple occurrence of the same subterm in a term, whereas the RTA force a unique subtree for all the occurrences of a rigid state. However, these classes of languages are incomparable: the language $L_0 = \{g(t,t) \mid t \in \mathcal{T}(\Sigma)\}$ is recognizable by a RTA (or positive TAGED) but $dag(L_0)$ is not DA recognizable. For the complement of this language, the situation is opposite – see [Jacquemard et al., 2011a] for a more detailed comparison.

	TA	RTA pos. TAGC[$\approx_{\mathsf{ref}}$]	TAGED[$\approx$] pos. TAGC[$\approx$]	DA
$\cup$	PTIME	PTIME	PTIME	PTIME
$\cap$	PTIME	EXPTIME	EXPTIME	not [ANR05]
$\neg$	EXPTIME	not	not	not
emptiness	linear-time	linear-time	EXPTIME-c.	NP-c.
membership	PTIME	NP-complete	NP-complete	NP-c.
$\cap$-emptiness	EXPTIME-c.	EXPTIME-c.	EXPTIME-c.	
universality	EXPTIME-c.	undecidable	undecidable	undecidable
inclusion	EXPTIME-c.	undecidable	undecidable	undecidable
finiteness	PTIME	PTIME	EXPTIME	

Table 3: Closure and decision results for RTA and related classes ([ANR05] is [Anantharaman et al., 2005]).

It was observed in the PhD thesis of Camille Vacher [Vacher, 2010] that for all

DA $\mathcal{D}$, one can build a positive TAGC[$\not\approx_{\mathsf{irr}}$] $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = dag^{-1}\big(\mathcal{L}_d(\mathcal{D})\big)$. Conversely, for all positive TAGC[$\not\approx_{\mathsf{irr}}$] $\mathcal{A}$, one can build a DA $\mathcal{D}$ such that $\mathcal{L}_d(\mathcal{D}) = dag\big(\mathcal{L}(\mathcal{A})\big)$. The latter direction involves a subset construction, hence the size of $\mathcal{D}$ is exponential in the size of $\mathcal{A}$. This can be summarized as follows.

Theorem 24 [Vacher, 2010]

Positive TAGC[$\not\approx_{\mathsf{irr}}$] and DA are equally expressive.

This result can be extended [Vacher, 2010] in order to handle global equality constraints, by the addition of a further restriction in the definition of runs of DA which is compatible with the NP emptiness decision procedure for DA of [Charatonik, 1999]. Alternatively, this can be defined[3] by adding to DA some global constraints of type $\approx$, interpreted on a run r on a t-dag d by $r \models q \approx q'$ if for all nodes p, p' of d such that $r(p) = q$ and $r(p') = q'$, then $p = p'$. Then, if DA[$\approx$] denote DA extended with such constraints, we have the following result.

Theorem 25

Positive TAGC[$\approx, \not\approx_{\mathsf{irr}}$] and DA[$\approx$] have the same expressiveveness.

The transformation from positive TAGC[$\approx, \not\approx_{\mathsf{irr}}$] to DA[$\approx$] is the same as above (hence it is exponential). Moreover, emptiness is still NP-complete for DA[$\approx$], and hence we solved the question of the decidability of the emptiness for TAGED (which are positive TAGC[$\approx, \not\approx_{\mathsf{irr}}$]).

Corollary 26

Emptiness is decidable in NEXPTIME for TAGED.

Another consequence is that for TAGED, it is not restrictive to assume that the binary relation on states defined by the disequalities of the global constraint is maximal (*i.e.* the global constraint is a conjunction $\bigwedge_{q \neq q'} q \not\approx q' \wedge C'$ where C' contains only atomic equality constraints. Note that it was already observed that it is not restrictive to assume that all the atomic equality constraints have the form $q \approx q$ (like for RTA).

Perspectives: DAG Compression with Partial Sharing

An alternative definition of DAG automata could be to consider DAGs representation of terms with partial sharing. For instance, one could impose a maximal sharing for sibling subtrees (and no sharing for the others). This is is an assumption of the Active

[3] Igor Walukiewicz, personal communication

XML model [Abiteboul et al., 2008] (note though that the AXML trees are unranked unordered labeled trees). It would be interesting to know whether the complexity of emptiness for automata running on such DAGs is better than for DA.

3.4 Boolean combinations of equalities and disequalities

For all the automata classes presented in Sections 3.1-3.3, the global constraints are restricted to be conjunctions of atomic equality and disequality constraints. We show with Luis Barguñó, Carlos Creus, Guillem Godoy, and Camille Vacher that this is not a real limitation (*i.e.* that conjunctions of atomic constraints are as expressive than Boolean combinations of atomic constraints).

Theorem 27 [Barguñó et al., 2010]

Positive $\mathsf{TAGC}[\approx, \not\approx_{\mathsf{irr}}, \not\approx_{\mathsf{ref}}]$ and $\mathsf{TAGC}[\approx, \not\approx_{\mathsf{irr}}, \not\approx_{\mathsf{ref}}]$ have the same expressiveness.

The proof of this result uses an intermediate class of constraints presented below in Section 3.5.
The main result of [Barguñó et al., 2010] is the decidability of emptiness for all TAGC.

Theorem 28 [Barguñó et al., 2010]

Emptiness is decidable for $\mathsf{TAGC}[\approx, \not\approx_{\mathsf{irr}}, \not\approx_{\mathsf{ref}}]$.

The proof is based on a rather involved pumping argument using a well quasi-ordering $\leq$. Given a positive $\mathsf{TAGC}[\approx, \not\approx_{\mathsf{irr}}, \not\approx_{\mathsf{ref}}]$ $\mathcal{A}$, we associate to a successful run r of $\mathcal{A}$ on a term t one measure e_i for each $0 \leq i \leq h(t)$. The measures are such that a pumping transforming the run r into a smaller run (*wrt* a well founded ordering) is possible as soon as there exists an increasing pair of elements $e_i \leq e_j$ in the sequence $e_1, \ldots, e_{h(t)}$ associated to t. By Higman lemma, Kruskal lemma and a condition on the first element and the relation between an element e_i and the next e_{i+1} in the sequences associated to the runs, we show that there exists a bound B on the height of runs with no possible pumpings, and hence that emptiness is decidable. The value of the bound B is not known explicitly, though.

The above emptiness decidability result still holds for more general automata models, like the extension of $\mathsf{TAGC}[\approx, \not\approx_{\mathsf{irr}}, \not\approx_{\mathsf{ref}}]$ with local brother equality and disequality constraints such as those of TAB described in Section 1.1, and also a similar automata model with global constraints computing on unranked trees.

In a long version of [Barguñó et al., 2010], currently in submission, the above results are furthermore generalized by allowing the equality and disequality constraints (both for global and the local constraints between brother subtrees) to be interpreted modulo a flat equational theory.

Perspectives: DA and TAGED

We are currently working with Anca Muscholl, Camille Vacher and Igor Walukiewicz in a generalization of the approach presented in Section 3.3, with constrained DA, extending DA[$\approx$] (which is equivalent to TAGED), and capturing the full model of TAGC. The goal is to express the constraints of type $\not\approx_{\mathsf{ref}}$, and to deduce an elementary emptiness decision procedure for TAGC[$\approx, \not\approx_{\mathsf{irr}}, \not\approx_{\mathsf{ref}}$].

3.5 Arithmetic Constraints

A feature for counting the number of occurrences of states is a natural extension of automata, which permits ones to deal with numerical properties of trees in input, useful for instance in the context of XML querying [Seidl et al., 2003, 2004; Lugiez, 2005; Dal Zilio and Lugiez, 2006; Seidl et al., 2008].

We consider with Luis Barguñó, Carlos Creus, Guillem Godoy, and Camille Vacher [Barguñó et al., 2010] the extension of TAGC with the following two kinds of counting constraints. A *linear inequality* over a state set Q is an expression of the form $\sum_{q\in Q} a_q \cdot |q| \geq a$ or $\sum_{q\in Q} a_q \cdot \|q\| \geq a$ where every a_q and a belong to $\mathbb{Z}$. When the above coefficients a_q and a have all the same sign, then the inequality is called a *natural linear inequality* over Q (note that it is equivalent to consider inequalities in both directions whose coefficients are all non-negative). The types of the linear and natural linear inequalities are denoted by $|.|_{\mathbb{Z}}$, $\|.\|_{\mathbb{Z}}$ and $|.|_{\mathbb{N}}$, $\|.\|_{\mathbb{N}}$.

The satisfaction of a linear equalities by a run r of a TA on a ranked term t is defined by the interpretation in integers, after replacing the expressions $|q|$ and $\|q\|$ by their respective interpretations as the following cardinalities: $[\![\, |q| \,]\!]_r = |r^{-1}(q)|$ and $[\![\, \|q\| \,]\!]_r = \big|\{t|_p \mid p \in \mathcal{P}os(t), r(p) = q\}\big|$.

The class TAGC[$|.|_{\mathbb{Z}}$] has been studied under different names, *e.g.* Parikh automata in [Klaedtke and Ruess, 2002], linear constraint tree automata in [Bojańczyk et al., 2009]... It has a decidable emptiness test. Indeed, the set of successful runs of a given TA with state set Q is a context-free language (seeing runs as words of Q^*), and the Parikh projection (the set of tuples over $\mathbb{N}^{|Q|}$ whose components are the $[\![\, |q| \,]\!]_r$ for every run r) of such a language is a semi-linear set. The idea for deciding emptiness for a TAGC[$|.|_{\mathbb{Z}}$] $\mathcal{A}$ is then to compute this semi-linear set and to test the emptiness of its intersection with the set of solutions in $\mathbb{N}^{|Q|}$ of the arithmetic constraint of $\mathcal{A}$ (a Boolean combination of linear inequalities of type $|.|_{\mathbb{Z}}$) which is also semi-linear. This can be done in NPTIME, see [Bojańczyk et al., 2009]. The global constraints of types $|.|_{\mathbb{Z}}$ and $\|.\|_{\mathbb{Z}}$ are also discussed in III.2.

The *Presburger automata* [Seidl et al., 2003, 2008] contain constraints for counting the siblings of unranked trees. These constraints are tested at every transition steps, hence, following the terminology of Section 1, they are *local constraints*.

Combining global constraints of type $\approx$ and counting constraints of type $|.|_{\mathbb{Z}}$ leads to undecidability.

Theorem 29 [Barguñó et al., 2010]

Emptiness is undecidable for positive $\mathsf{TAGC}[\approx, |.|_{\mathbb{Z}}]$.

The proof is based on a reduction from the Hilbert's tenth problem. The main idea is that we can encode (non negative) integer multiplication by combining $\approx$ and $|.|_{\mathbb{Z}}$.

The main difference between the linear inequalities of type $|.|_{\mathbb{Z}}$ and $|.|_{\mathbb{N}}$ (and respectively $\|.\|_{\mathbb{Z}}$ and $\|.\|_{\mathbb{N}}$) is that the former can compare the respective number of occurrences of two states, like *e.g.* in $|q| \leq |q'|$, whereas the latter can only compare the number of occurrences of one state (or a sum of the number occurrences of several states with coefficients) to a constant as *e.g.* in $|q| \leq 4$ or $|q| + 2.|q'| \leq 9$. This difference permits us to obtain the decidability of the extension of TAGC with arithmetic constraints.

Theorem 30 [Barguñó et al., 2010]

$\mathsf{TAGC}[\approx, \not\approx, |.|_{\mathbb{N}}, \|.\|_{\mathbb{N}}]$ and positive $\mathsf{TAGC}[\approx, \not\approx]$ have the same expressiveness.

Actually, the arithmetic constraints are used in [Barguñó et al., 2010] as an intermediate formalism in order to eliminate the negations of equality and disequality atoms in the global constraint of a TAGC. Intuitively, the negations are replaced using some copies of the states and tests on the number of occurrences of states in runs, and of the number of subterms accepted in some states in runs. With Theore 28, we have that

Corollary 31 [Barguñó et al., 2010]

Emptiness is decidable for $\mathsf{TAGC}[\approx, \not\approx, |.|_{\mathbb{N}}, \|.\|_{\mathbb{N}}]$.

3.6 Application to the Static Analysis of XML Specifications

Semi-structured data, unlike relational tables, are not supposed to respect a predefined schema in order to be interpreted. They are rather labeled graphs, and data is contained not only in the labels but also in the structure. In the case of XML documents, the only requirement is well-formedness, meaning essentially that opening and closing tags are well parenthesized. This condition makes well-formed documents amenable to a labeled tree representation. This flexibility makes XML a uniform data model, which has developed as a standard for the exchange of data on the Web.

Some constraints can be expressed in order to impose more structure to XML documents. For instance, typing formalisms like DTDs or XML Schema permit ones to constrain how the tags are arranged in trees. The typing formalisms in use are all subcases of tree automata. Other constraints include variants of the integrity

constraints found in relational databases [Abiteboul et al., 1995], from which many XML documents are generated, like inclusions, keys and foreign keys.

Tree automata with global constraints are a good tool for reasoning about combinations of typing and integrity constraints: the typing constraints can be expressed as tree automata, and the integrity constraints can be expressed as global constraints. They permit ones to decide properties like *consistency* (does there exists at least one document satisfying a given set of constraints) of *entailment* (do some constraints imply others). Applications of these properties include data integration, query optimization or database normalization. We will discuss in the next paragraphs the interest of the above tree automata models in this context.

The logics on trees have also attracted a lot of attention in the context of XML foundations, as they provide a basis for query languages and for verification problems. We will discuss also below the extension to tree automata with constraints of the well-known connection between tree automata and monadic second order logic on labeled trees. In Part II, we will consider the question of the static analysis of XML transformations.

XML Type Definitions, Unranked Tree Recognizers

Type definitions can be provided along with XML documents in order to constrain their structure. They also provide users some guidance for accessing them through query languages. There are many formalisms for defining type of XML documents, with different expressive power. The less expressive and also the most frequently used are the DTDs. A DTD over Σ can be defined as a function D mapping Σ to regular expressions over Σ. The popular type definition language Relax NG, has the same expressive power as tree automata. XML Schema is argued to have an intermediate expressiveness [Murata et al., 2005]. We won't describe all the type definition formalisms here, but we shall just remark that regular tree languages capture them all – see *e.g.* [Murata et al., 2005] who presents several XML schema languages and their corresponding subclasses of regular tree grammars.

There is however an important difference between the tree automata corresponding to XML type definitions and the tree automata considered so far in this document. Indeed, the common abstraction of well-formed XML documents is unranked ordered trees (the number children of a position is not bounded) whose positions are labeled by symbols from a finite alphabet [Vianu, 2001; Schwentick, 2007], whereas the automata presented in the above sections recognize sets of ranked terms, whose labeling respect the arities specified in the signature. The *hedge automata* [Murata, 2000] are recognizers of unranked ordered labeled trees generalizing the above tree automata computing on ranked terms. They are presented formally in Section II.2.1 below. A run of an hedge automata is still a function from the set of position of an unranked tree into states, and hence the definition of the satisfiability of the above global constraints can be extended naturally from ranked terms to unranked ordered trees.

The above emptiness decision results for TAGC carry over to their generalization to unranked trees, using a standard bijection from unranked ordered trees to ranked binary terms like the one presented in Section II.2.1 below, and corresponding transformations of hedge automata into (binary) tree automata.

The generalization of tree automata with local constraints from ranked to unranked trees is not so immediate. Some models have been proposed for instance for generalizing TAB to unranked ordered trees [Wong and Löding, 2007], [Löding and Wong, 2009], and this topic is discussed in Part III, Section 2.4.

Integrity Constraints

In addition to a type definition, one may want to impose some integrity constraints to XML documents. The notion of keys is traditionally used in database theory in order to characterize a unique identifier for an element of data [Abiteboul et al., 1995]. The *key* constraint [Buneman et al., 2001]

$$q[\pi_1, \ldots, \pi_k] \to q \qquad \text{(key)}$$

expresses that every two positions p and p' of type q coinciding on attributes $\pi_1, \ldots, \pi_k$ must be the same. The above key is called *unary* if $k = 1$.

For simplicity, we assume that every *attribute* π_i is a *simple path* (a sequence of labels), which, given a position p, defines the set of positions reachable from p following the path π_i. There are several possible definitions of the type q of a position p: it can be for instance an *element* type of a DTD (*i.e.* the label of the position p), or a *path expression* π (p can be reached from the root of the tree following a path defined by π), or more generally a state of a tree automaton (it is well known that XPath can be translated into tree automata, see *e.g.* [Francis et al., 2011]).

A *foreign key* constraint is the conjunction of an *inclusion* constraint and a key constraint [Buneman et al., 2001], such as the following

$$q[\pi_1, \ldots, \pi_k] \subseteq q'[\pi'_1, \ldots, \pi'_k],\ q'[\pi'_1, \ldots, \pi'_k] \to q' \qquad \text{(fkey)}$$

The above inclusion constraint expresses that for every position p of type q, there exists a position p' of type q' such that the attributes $\pi_1, \ldots, \pi_k$ of p coincide with the attributes $\pi'_1, \ldots, \pi'_k$ of p'. Inclusion constraints, functional constraints and inverse constraints can also be defined in term of path expressions [Fan and Siméon, 2000].

The above notion of attribute that coincide was purposely left unspecified and may actually have several meanings. Remember that the attributes of a position p are positions reachable from p following some given paths. A first definition (att_1) refers to a model called *data trees* where every position in a tree representing an XML document carries not only a label in a finite alphabet but also a *data value* from an infinite domain. In this case, coincidence of attributes positions is simply the equality of the respective values at these positions. A stronger definition (att_2) of coincidence of attributes positions is the equality of these positions in the tree. At last, an even stronger definition (att_3) is based on the subtree equality, *i.e.* the

equality constraints of the above sections: two attribute positions π_1 and π_2 coincide if the respective subtrees at π_1 and π_2 are isomorphic. Note that if we use a DAG representation of tree with maximal sharing like in Section 3.3, then the definitions (att_1) and (att_2) are equivalent, but in general, (att_2) is a particular case of (att_3). The relation between the notions (att_1) and (att_3) is further discussed as a perspective in Part III, Section 2.

The approach (att_3) for equality corresponds to the idea that there is no distinction between the structure of the tree and the data – this kind of flexibility is considered as a strong point of XML and semi-structured data. In this settings, the above TAGC, generalized to unranked ordered trees, offer a uniform framework for defining combinations of type definitions and integrity constraints. For instance, positive $\mathsf{TAGC}[\not\approx_{\mathsf{ref}}]$ can express unary key constraints in presence of a regular type definition, *wrt* the definition (att_3) of attribute equality. Indeed, if we define a state q_1 characterizing the positions corresponding to attribute π_1 (*i.e.* every position p_1 reached from a position p of type q following π_1), then the constraint $q_1 \not\approx q_1$ corresponds to the above key constraint (key), when $k = 1$ (the subtrees at two different positions in state q_1 are not isomorphic).

Similarly, the positive $\mathsf{TAGC}[\not\approx_{\mathsf{irr}}]$ can express *denial constraints* of the form

$$q[\pi_1 \ldots, \pi_k] \neq q'[\pi_1 \ldots, \pi_k].$$

A constraint as above says that two positions of respective type q and q' cannot coincide on attributes $\pi_1 \ldots, \pi_k$. Such constraint can be used to deal with inconsistent data.

Reasoning on combinations of integrity constraints in presence of a type definition is known to be difficult. For instance, in the above case (att_1), the problem of consistency of a given (unranked ordered) tree automaton $\mathcal{A}$ and a set of unary key and foreign key constraints (whether there is an XML document in $\mathcal{L}(\mathcal{A})$ satisfying the constraints) is NP-complete and it is undecidable for DTDs and general keys and foreign keys [Fan and Libkin, 2002].

Perspective: Complexity Bounds for TAGC

In the case (att_3), consistency reduces to emptiness decision for TAGC, hence a precise complexity analysis of this problem is interesting in this setting.

For the full TAGC class, (Theorems 28 and Corollary 31) the current upper bounds are non-elementary. As explained in Section 3.4, finding an elementary upper bound is the subject of an ongoing work. The best lower bound known for TAGC emptiness decision is EXPTIME-hardness [Filiot et al., 2008].

The upper bounds for emptiness decision of subclasses of TAGC presented above are, to sum up, NEXPTIME for TAGED (*i.e.* positive $\mathsf{TAGC}[\approx, \not\approx_{\mathsf{irr}}]$), EXPTIME for positive $\mathsf{TAGC}[\approx]$ and linear time for RTA. The exact complexity of emptiness decision for $\mathsf{TAGC}[\not\approx_{\mathsf{ref}}]$ is EXPTIME-completeness[4]. The latter result is good to

[4]Guillem Godoy, personal communication.

know in the context of consistency of a given tree automaton with a set of unary XML keys. This indicates in particular that this problem seems harder for (att_3) than for (att_1). Note however that the above integrity constraints, interpreted in the case (att_3), are translated into TAGC such that the test positions, for equality and disequality tests, are pairwise incomparable (*wrt* prefix ordering). It would be interesting also to investigate the complexity of emptiness decision for TAGC in this restricted case.

Monadic Second Order Logic

Monadic second order logic (MSO), when interpreted on the positions of labeled trees, provides a formalism for Web data querying, which is considered as appropriate in the sense that it has a good expressiveness and its satisfiability is decidable. By querying, we mean the extraction of tuples of subtrees from a given XML tree. The arity of a query is the length of the tuple extracted, defined by the number of free variable in a MSO formula.

The seminal result of [Thatcher and Wright, 1968] states that MSO has exactly the same expressiveness as TA, and therefore its satisfiability is decidable, although with an non-elementary complexity, see also *e.g.* [Thomas, 1997], [Courcelle, 1990] for the connections between logic and automata.

Several (ranked or unranked) tree automata based XML query mechanisms capturing the power of MSO have been proposed in [Neumann and Seidl, 1998], [Neven and Schwentick, 2002] for unary queries and [Berlea and Seidl, 2004], [Niehren et al., 2005] for n-ary queries. Monadic Datalog is also shown equivalent to MSO for unary queries [Gottlob and Koch, 2004]. MSO has been extended *e.g.* with numerical constraints [Seidl et al., 2003, 2008], and by attribute grammars in [Neven and Van den Bussche, 2002].

The TAGEDs have been introduced [Filiot et al., 2007] as a decision tool for the satisfiability of a guarded fragment of TQL [Cardelli and Ghelli, 2004], which is a spatial logic defined for querying unranked ordered labeled trees, with spatial predicates which are algebraic operation on hedges (composition into tree (addition of root), concatenation), a fixpoint operator and tree variables. It is close to XML pattern-matching languages [Benzaken et al., 2003], and permits ones in particular to define pattern with tree variables (matched against XML trees) and context variables (matched against n-ary contexts). It is possible to have multiple occurrences of the same tree variable x in a pattern, expressing tree isomorphism. As a consequence, TQL is more expressive than MSO. Moreover, in a pattern of the form $X(\mathsf{id}(x), \mathsf{id}(x))$, where X is a binary context variable, the two occurrences of the tree variable x may be arbitrary distant. Hence, a tree automaton recognizing the instance of this pattern cannot use a local constraint in order to test the equality of the two instances of x.

The monadic second order formulae are built with the usual Boolean connectors, with quantifications over first order variables (interpreted as positions), denoted $x, y \ldots$ and over unary predicates (*i.e.* monadic second order variables interpreted as sets of positions), denoted $X, Y \ldots$, and with the following predicates, whose inter-

pretation domain is the set of positions $\mathcal{P}os(t)$ of a ranked term $t \in \mathcal{T}(\Sigma)$: equality (between positions) $x = y$, membership $X(x)$ (the position x belongs to the set X), labeling $a(x)$, for $a \in \Sigma$ (the position x is labeled by a in t), and navigation $S_i(x, y)$, for all i smaller than or equal to the maximal arity of symbols of Σ (y is the ith child of x in t). If we want to interpret the formulae on unranked ordered trees, the above successor predicates $S_i(x, y)$ are not sufficient, and are replaced by: $S_{\downarrow}(x, y)$ (y is a child of x), and $S_{\rightarrow}(x, y)$ (y is the immediate successor sibling of x). Note that the above predicates $S_1, S_2, \ldots$ can be expressed using these two predicates only.

Some strict extensions of MSO with predicates corresponding to the above equality, disequality and arithmetic constraints have been proposed in [Filiot et al., 2008] and then generalized in [Barguñó et al., 2010]. The new predicates for term equality $X \approx Y$, and term disequality $X \not\approx Y$ hold when for all p in X and all p' in Y, $t|_p = t|_{p'}$, resp. $t|_p \neq t|_{p'}$, (we note these predicate types $\approx$ and $\not\approx$). We consider also [Barguñó et al., 2010] *linear inequalities*: $\sum a_i.|X_i| \geq a$ or $\sum a_i.\|X_i\| \geq a$, where a_i and a belong to $\mathbb{Z}$ ($|X_i|$ is interpreted as the cardinality of X_i and $\|X_i\|$ as the cardinality of $\{t|_p \mid p \in X_i\}$). The corresponding predicate types are denoted $|.|_{\mathbb{Z}}$, $\|.\|_{\mathbb{Z}}$, and $|.|_{\mathbb{N}}$ and $\|.\|_{\mathbb{N}}$ for the restriction where all the coefficients are of the same sign.

We write $\mathsf{MSO}[\tau_1, \ldots, \tau_k]$ for the set of monadic second order logic formulae with equality, membership, labeling predicates and other predicates of type $\tau_1, \ldots, \tau_k$, amongst the above types $\approx$, $\not\approx$, and $|.|_{\mathbb{N}}$, $\|.\|_{\mathbb{N}}$ and $+1$, this latter denoting the successor predicates. Let $\mathsf{EMSO}[\tau_1, \ldots, \tau_k]$ be the fragment of $\mathsf{MSO}[\tau_1, \ldots, \tau_k]$ containing the formulae of the form $\exists X_1 \ldots \exists X_n\, \phi$ such that all the atoms of type $\approx$, $\not\approx$, $\mathbb{Z}$ or $\mathbb{N}$ in ϕ involve only second order variables amongst $X_1, \ldots, X_n$.

The theorem of Thatcher and Wright [Thatcher and Wright, 1968] states that $\mathsf{MSO}[+1]$ is decidable because it has the same expressiveness as TA. The extension $\mathsf{MSO}[+1, \approx]$ is undecidable, see *e.g.* [Filiot et al., 2007]. The extension $\mathsf{MSO}[+1, |.|_{\mathbb{Z}}]$ is undecidable as well [Klaedtke and Ruess, 2002].

On the other side, the fragment $\mathsf{EMSO}[+1, |.|_{\mathbb{Z}}]$ is decidable [Klaedtke and Ruess, 2002], and a fragment of $\mathsf{EMSO}[+1, \approx, \not\approx]$ is shown decidable in [Filiot et al., 2008] for a restricted variant of $\not\approx$, using a two way correspondence between these formulae and a decidable subclass of TAGED. This latter construction has been straightforwardly adapted in order to establish a two way correspondence between $\mathsf{EMSO}[+1, \approx, \not\approx, \mathbb{N}]$ and $\mathsf{TAGC}[\approx, \not\approx, |.|_{\mathbb{N}}, \|.\|_{\mathbb{N}}]$.

Theorem 32 [Barguñó et al., 2010]

$\mathsf{EMSO}[+1, \approx, \not\approx, |.|_{\mathbb{N}}, \|.\|_{\mathbb{N}}]$ is decidable.

II

Verification of Infinite State Systems

Significant progress in the areas of specification and verification of systems have been achieved through *model-checking* [Vardi and Wolper, 1986], an algorithmic method for exploring exhaustively the set of reachable configurations (assumed finite) of a system, in order to verify that the system's behavior meets certain properties expressed by logical formulas. The use of these techniques is fairly widespread, in particular through symbolic methods for representing very large (but finite) sets of configurations. An important current research topic deals with the generalization of these techniques to systems with an infinite set of reachable configurations, for instance when the configurations contain a finite but unlimited amount of information and they can be represented by words or more generally by finite trees. *Regular model checking* [Wolper and Boigelot, 1998; Abdulla et al., 2002] is a general framework for reasoning about such systems. It uses a finite representation of infinite sets of configurations by automata on words or trees. The dynamics of the systems (the transition between the system's configurations) are represented by abstract reduction systems like transducers or rewriting systems, and reasoning on reachability properties of such systems can then be reduced to the computation of the transitive closure of automata languages by the transducer or rewrite system. This problem can be summarized by the following general property, that we simply call *model checking* below: given an automaton accepting a language L_{in}, which represents a set of possible initial configurations of the system, another automaton accepting a language L_{err}, which represents a set of erroneous configurations, and given a binary relation $\mathcal{R}$ (defined, say, by a rewrite system), we want to decide whether the rewrite closure of L_{in} by $\mathcal{R}$, denoted $post^*_{\mathcal{R}}(L_{\mathsf{in}})$, has an empty intersection with L_{err}. Note that non-*reachability*, the problem whether a given target configuration t cannot be reached by $\mathcal{R}$ starting from a given source configuration (s) is a particular case of this problem, when $L_{\mathsf{in}} = \{s\}$ and $L_{\mathsf{err}} = \{t\}$.

When (in the context of tree configurations) the transitive closure $post^*_{\mathcal{R}}(L_{\mathsf{in}})$ is the language of a TA, and this TA can effectively constructed. then model checking reduces to a decision problem for TA, namely the emptiness of intersection. Therefore, the problem of the preservation of regularity of tree languages by rewrite rules (proving that the rewrite closure of a TA language is again a TA language) has deserved quite some attention. For restricted classes of rewrite rules, defined by syntactical restrictions, the result can be established with a procedure of completion of the TA for

L_{in}, adding some tree automata transitions by superposition of the rewrite rules into the the TA transitions rules, see *e.g.* [Genet and Rusu, 2010] for a recent reference. In some other cases, finding an effective TA construction requires the use of accelerations techniques, *e.g.* [Bouajjani and Touili, 2002] or [Jacquemard and Rusinowitch, 2008a][1]. We present several results around these problems in the case of term rewriting systems (TRS) (applying to ranked trees) in Section 1.

The regular model checking is also related to *static typechecking* programs for tree transformations, see [Milo et al., 2003], and the beginning of Section 2, which is the problem to verify that a program always converts valid source trees (semi-structured documents) into valid output trees (where types are defined by TA). This problem concerns in general unranked trees. In Section 2, we consider generalizations of automata models for unranked trees (hedge automata and variants) and term rewriting systems for unranked trees (hedge rewrite systems), and we present results on the rewrite closures of (unranked) tree languages. We will see in particular in Section 2.5 that this can be applied to the verification of consistency of read/write access control policies to XML documents.

In general, the transitive closure $post^*_{\mathcal{R}}(L_{\mathsf{in}})$ is not a regular tree language, and we are left with several other options. For instance, it is possible to define a safe over-approximation of $post^*_{\mathcal{R}}(L_{\mathsf{in}})$ by a TA [Genet and Rusu, 2010]. Alternatively, it might also be the case that the model checking is undecidable, but the membership to $post^*_{\mathcal{R}}(L_{\mathsf{in}})$ is decidable, hence in particular reachability is decidable in this case, we consider similar situations in Section 1.4. It can occur also if $post^*_{\mathcal{R}}(L_{\mathsf{in}})$ is a context-sensitive language. In general, one can consider more expressive, yet decidable, formalisms for defining tree language, in order to capture $post^*_{\mathcal{R}}(L_{\mathsf{in}})$, as well as its intersection with L_{err}. We present some work following this approach in Section 2 (using so called context-free hedge automata), Section 3.1 (using TAB), Section 3.2 (with context-free and context-sensitive ranked tree languages), and we discuss the problem of the definition of context-free unranked tree languages in this context, in Section 3.3.

1 Term Rewriting

Term rewriting systems (TRS) and the rewrite relations are defined in Section I.1. Given a TRS $\mathcal{R}$ over a signature Σ and $L \subseteq \mathcal{T}(\Sigma)$, we note $pre^*_{\mathcal{R}}(L) = \{s \mid \exists t \in L, s \xrightarrow[\mathcal{R}]{*} t\}$, $post^*_{\mathcal{R}}(L) = \{t \mid \exists s \in L, s \xrightarrow[\mathcal{R}]{*} t\}$, and $\mathsf{NF}_{\mathcal{R}}(L) = post^*_{\mathcal{R}}(L) \cap \mathsf{NF}_{\mathcal{R}}$.

Some terms $s_1, \ldots, s_n$ are called *joinable* by $\mathcal{R}$ (resp. they have a *common ancestor wrt* $\mathcal{R}$) if there exists a term t such that $s_i \xrightarrow[\mathcal{R}]{*} t$ (resp. $t \xrightarrow[\mathcal{R}]{*} s_i$) for all i in $\{1, \ldots, n\}$.

A rewrite rule $\ell \to r$ is called *left-ground* (resp. *right-ground*, *ground*) if $\ell \in \mathcal{T}(\Sigma)$ (resp. $r \in \mathcal{T}(\Sigma)$, $\ell, r \in \mathcal{T}(\Sigma)$), *left-linear* (resp. *right-linear*, *linear*) if ℓ (resp. r, both) is linear, and similarly for *flat* and *shallow*. It is called *collapsing* if $r \in vars(\ell)$. A TRS $\mathcal{R}$ is called ground (linear, *etc*) if all its rules are ground (linear, *etc*).

[1] the later result is presented in Section 2.2.

In this section, we present some studies of the closure of ranked terms languages under ranked term rewriting. We wont follow the organizations of the other sections, and will present some applications first, with examples used to motivate the following theoretical results. Some of these examples extend the cases presented in Introduction.

Analysis of Functional Programs

Functional programs manipulating tree structured data values with pattern matching can be described by rewrite rules [Jones and Andersen, 2007] such that the rewriting relation represents the program evaluation. Let us consider for instance the TRS containing the two following rules, which define the operators app for the concatenation of two lists and rev for the reverse of one list ([Genet and Tong, 2001]),

$$\begin{array}{rcl} \mathsf{app}(\mathsf{nil}, y) & \to & y, \\ \mathsf{app}\big(\mathsf{cons}(x, y), z\big) & \to & \mathsf{cons}\big(x, \mathsf{app}(y, z)\big), \\ \mathsf{rev}\big(\mathsf{cons}(x, y)\big) & \to & \mathsf{app}\big(\mathsf{rev}(y), \mathsf{cons}(x, \mathsf{nil})\big) \end{array}$$

and let us consider the set L_{in} of initial "values", containing terms of the form

$$\mathsf{rev}(\mathsf{cons}(0, \ldots, \mathsf{cons}(0, \mathsf{cons}(1, \ldots, \mathsf{cons}(1, \mathsf{nil})))))$$

where 0 and 1 are constant function symbols. Note that L_{in} is a regular tree language. An analysis of the terms of $post^*_{\mathcal{R}}(L_{\mathsf{in}})$, which is too long to be performed by hand, shows that this closure is a regular tree language. An example of the construction of a tree automaton for $post^*_{\mathcal{R}}(L_{\mathsf{in}})$, obtained with the system Timbuk, is presented in [Genet and Tong, 2001]. The use of some accelerations (defined by the user) is required during the construction. Moreover, with a second tree automata construction, it can be checked that the intersection of the closure of L_{in} with the set $\mathsf{NF}_{\mathcal{R}}$ (this latter set is also a regular tree language because $\mathcal{R}$ is left-linear) contains exactly the terms of the form

$$\mathsf{cons}(1, \ldots, \mathsf{cons}(1, \mathsf{cons}(0, \ldots, \mathsf{cons}(0, \mathsf{nil})))).$$

This shows that the values returned by the program on the above set of inputs (*i.e.* the normal forms obtained from the terms of L_{in}) are as expected.

This approach based on tree automata construction enables reachability analysis and flow analysis of functional programs. For instance, in [Jones and Andersen, 2007], regular tree grammars are constructed in order to approximate the *collecting semantics* of programs (roughly a mapping associating to each program point the set of configurations reachable at that point), when a program is represented by a left-linear TRS as above. In [Kochems and Ong, 2011], a finer approximation is refined by using indexed linear tree grammars instead of regular grammars.

Analysis of Networks with a Tree Architecture

A similar approach can be applied to the analysis of communicating processes organized in network with a tree-like architecture [Abdulla et al., 2002; Bouajjani and Touili, 2002; d'Orso and Touili, 2006]. Case studies include leader election protocols, mutual exclusion protocols (like the tree arbiter protocol) and multicast protocols.

The goal of such analysis is to be applicable to networks of any size, in other terms, the analysis is performed on a system parametrized by the number of processes, with an infinite number of instances. Hence the number of configurations to consider is infinite, but as in the cases cited above, it is a regular tree language.

In the case studies cited above, the communications typically consist in the exchange of a token between positions and children or parents positions, and can be represented by rules of tree transducers, like the rules presented in Figure 7, which model a token tree protocol [Abdulla et al., 2002].

Figure 7 Bottom-up tree transducer rules for a token tree protocol [Abdulla et al., 2002].

$$
\begin{array}{rcl}
\underline{\mathsf{n}} & \to & q_0(\underline{\mathsf{n}'}) \\
\mathsf{n}\big(q_0(x_1), q_0(x_2)\big) & \to & q_0\big(\mathsf{n}(x_1, x_2)\big) \\
\mathsf{n}\big(q_1(x_1), q_0(x_2)\big) & \to & q_2\big(\mathsf{t}(x_1, x_2)\big) \\
\mathsf{n}\big(q_2(x_1), q_0(x_2)\big) & \to & q_2\big(\mathsf{n}(x_1, x_2)\big)
\end{array}
\qquad
\begin{array}{rcl}
\underline{\mathsf{t}} & \to & q_1(\underline{\mathsf{n}'}) \\
\mathsf{t}\big(q_0(x_1), q_0(x_2)\big) & \to & q_1\big(\mathsf{n}(x_1, x_2)\big) \\
\mathsf{n}\big(q_0(x_1), q_1(x_2)\big) & \to & q_2\big(\mathsf{t}(x_1, x_2)\big) \\
\mathsf{n}\big(q_0(x_1), q_2(x_2)\big) & \to & q_2\big(\mathsf{n}(x_1, x_2)\big)
\end{array}
$$

Note that the rules of Figure 7 form a TRS $\mathcal{R}$ over a signature containing the binary symbols t and n, for inner positions with, resp. without, the token, the nullary symbols $\underline{\mathsf{t}}$, $\underline{\mathsf{t}'}$ and $\underline{\mathsf{n}}$, $\underline{\mathsf{n}'}$ for the leaf positions (with the same signification) and three unary *state* symbols: q_0 (the token is neither on the position nor on a position below) q_1 (the position is releasing the token to its parent's position) and q_2 (the token is either in the position or below).

It can be shown that given regular tree language L_{in}, $post^*_{\mathcal{R}}(L_{\mathsf{in}})$ is regular (see [Seki et al., 2002] for a systematic study of TRS of the above kind, called *layered transducing* TRS therein). This permits ones to show in particular that if L_{in} is the subset of terms of $\mathcal{T}\big(\{\mathsf{t}, \mathsf{n}, \underline{\mathsf{t}}, \underline{\mathsf{n}}\}\big)$ containing exactly one token, then the intersection of $post^*_{\mathcal{R}}(L_{\mathsf{in}})$ with the set $\{q_2(t) \mid t \in \mathcal{T}\big(\{\mathsf{t}, \mathsf{n}, \underline{\mathsf{t}}, \underline{\mathsf{n}}\}\big),\ t$ contains at least 2 tokens$\}$ (this set is regular) is empty. This shows that the above token protocol ensures mutual exclusion property on every network.

Analysis of Imperative Programs

Tree automata techniques have also been used in many works for the analysis of imperative programs with recursive procedure calls and spawning of concurrent threads. Such programs can be modeled by ground term rewriting rules of two main kinds:

- pushdown rules, containing only constant symbols and the binary function symbol $\cdot$ which is associative (A) and represents the sequential composition,
- multiset rules, containing only constant symbols and the binary function symbol $\|$ which is associative and commutative (AC) and represents the parallel composition.

We assume a constant symbol 0, which is neutral *wrt* $\cdot$ and $\|$, representing the null process and some other process constant symbols $\mathtt{a}, \mathtt{b}, \ldots$ Figure 8 describes

Figure 8 A procedure `a` calls in (r_1) a second procedure `b`. The procedure `b` can either return true (`t`, with r_2) or false (`f`, with r_3). If `b` returns true, then a new thread is launched, containing a procedure `c` (r_4). If `b` returns false, then the procedure `a` returns (r_5) ([Bouajjani and Touili, 2002]).

```
void a() {
  while(true) {
    if b() {
    thread_create(&t1,c)
    } else { return }}}
```

$$\begin{array}{rcll} \mathtt{a} & \to & \mathtt{b}\cdot\mathtt{a} & (r_1)\\ \mathtt{b} & \to & \mathtt{t} & (r_2)\\ \mathtt{b} & \to & \mathtt{f} & (r_3)\\ \mathtt{t}\cdot\mathtt{a} & \to & \mathtt{a}\,\|\,\mathtt{c} & (r_4)\\ \mathtt{f} & \to & 0 & (r_5) \end{array}$$

for instance 5 rewrite rules [Bouajjani and Touili, 2002] representing an imperative program with a loop calling a procedure `b` and creating threads `c`. Note that (r_1) in Figure 8 calls `b` in a loop (`a` is restarted if `b` has returned with true) hence the number of thread created is unbounded.

A procedure call can be represented by a rule of the form $a \to b \cdot a'$: at program point a, a second procedure is called, which starts at program point b. When this second procedure returns, the program continues at program point a'. We can have $a' = a$, meaning that the second procedure b is called in a while loop. If we want moreover to represent a global state of the program (describing for instance the current state of some global variables with a finite data domain), we can add a new constant symbol at the beginning of the prefix sequence, see *e.g.* [Schwoon, 2002]. For instance, the rules for procedure calls will have the form $q \cdot a \to q' \cdot b \cdot a'$ (they corresponds to push transition of a pushdown automaton, the first symbol is the state of the automaton, the others represent the top of the stack). For a procedure return we have $q \cdot a \to q'$ (pop transitions of pda) and we can have also rule changing only the global state $q \cdot a \to q' \cdot a$ (internal transitions of pda).

The pushdown rules have the general form $a_1 \cdot \ldots \cdot a_n \to b_1 \cdot \ldots \cdot b_m$.

For a proper representation of sequential execution of programs, we consider a relation called *prefix rewriting* which differs from the rewrite relation defined above (for instance, we do not want to allow a rewriting of the form $a \to b{\cdot}a \to b{\cdot}b{\cdot}a$). Prefix rewriting associated to a rewrite system $\mathcal{R}$ as above is the smallest binary relation $\mapsto_{\mathcal{R}}$ containing the rules of $\mathcal{R}$ and such that if $s \mapsto_{\mathcal{R}} t$, then for all t', $s{\cdot}t' \mapsto_{\mathcal{R}} t{\cdot}t'$ and $s\|t' \mapsto_{\mathcal{R}} t\|t'$. Algorithms computing regular reachability sets of pushdown rewrite

systems (*wrt* prefix rewriting) have been proposed in [Caucal, 1992; Bouajjani et al., 1997].

The creation of a new thread can be represented by a rule of the form $a \to a \parallel b$. The multiset rules have the general form $a_1 \| \ldots \| a_n \to b_1 \parallel \ldots \parallel b_m$ and correspond to Petri net transitions, which were proposed for the representation of multithreaded programs (with synchronization) see *e.g.* [Delzanno et al., 2002].

PA processes are set of pushdown and multiset rules the left-hand sides of which are restricted to a single constant. In [Lugiez and Schnoebelen, 2002] it is shown that sets of representative (modulo A for $\cdot$ and AC for $\|$) of reachable terms can be characterized by ranked TA, hence that TA can be used for the reachability analysis of PA processes.

Process rewrite systems (PRS) are arbitrary combinations of pushdown rules and multiset rules. In [Bouajjani and Touili, 2005], the computation of an exact tree automata-based representation of the reachability set is proposed for restricted PRS whose multiset rewrite rules are preserving semilinear sets. This is the case for instance of the form $a \to b_1 \parallel \ldots \parallel b_m$. Approximations are proposed for arbitrary PRS.

For the related model of *dynamic pushdown networks*, reachability analysis can also be performed though reduction to tree automata decision problems [Seidl, 2009].

The close connections between regular set of terms modulo A for $\cdot$ and AC for $\|$ and unranked ordered, respectively unranked unordered trees are discussed in Sections 2.1 and 2.6.

1.1 Ground Term Rewriting Systems

It is undecidable in general whether a given TRS is preserving regularity [Gilleron, 1991]. A lot of efforts has been put into identifying classes of TRS, generally defined by syntactical restrictions, enjoying this property.

The first such class of TRS identified, and perhaps the simplest, is the class of ground TRS [Brainerd, 1969]. Given a TA $\mathcal{A}_{\mathsf{in}}$ and a ground TRS $\mathcal{R}$, it is not difficult to construct a TA recognizing $post^*_{\mathcal{R}}\big(\mathcal{L}(\mathcal{A}_{\mathsf{in}})\big)$. In a preliminary step, we construct a TA $\mathcal{A}_0$, obtained form $\mathcal{A}_{\mathsf{in}}$ by adding one new state q_r for each strict subterm of a *rhs* of rule of $\mathcal{R}$, and adding all the transitions of the form $f(q_{r_1}, \ldots, q_{r_n}) \to q_r$ such that $r = f(r_1, \ldots r_n)$. Hence, for all r, $\mathcal{L}(\mathcal{A}_0, q_r) = \{r\}$. Then, we iterate the following completion operation ($i \geq 0$): for each ground rule $\ell \to f(r_1, \ldots r_n)$ in $\mathcal{R}$ (with $n \geq 0$), and for each state q such that $\ell \in \mathcal{L}(\mathcal{A}_i, q)$, the TA $\mathcal{A}_{i+1}$ is obtained form $\mathcal{A}_i$ by adding the transition $f(q_{r_1}, \ldots, q_{r_n}) \to q$. No state is added in the above step, hence the number of transitions that can be added is finite and polynomial, and the construction will terminate in polynomial time with a TA $\mathcal{A}_k$ of size polynomial in the sizes of $\mathcal{A}_{\mathsf{in}}$ and $\mathcal{R}$, and such that $\mathcal{L}(\mathcal{A}_k) = post^*_{\mathcal{R}}\big(\mathcal{L}(\mathcal{A}_{\mathsf{in}})\big)$.

The stronger property holds that for every ground TRS $\mathcal{R}$, the rewrite relation $\xrightarrow{*}_{\mathcal{R}}$ is regular [Dauchet et al., 1987] – see Section I.2.5 for the definition of regular binary relations on ground terms. If a binary tree relation is regular, then the closure of every regular tree language under this relation is also regular.

1.2 Flat and Shallow Term Rewriting Systems

It is shown in [Nagaya and Toyama, 2002] that for every right-linear and right-shallow TRS $\mathcal{R}$ and every regular tree language L_{in}, the closure $post^*_{\mathcal{R}}(L_{\mathsf{in}})$ is regular. This regularity preservation results generalizes many former ones: for linear and right-flat (also called *monadic*) TRS [Salomaa, 1988], for linear and right-shallow (also called *semi-monadic* or *inverse-growing*) TRS [Coquidé et al., 1994; Jacquemard, 1996]...

The above regularity preserving have been further generalized to the classes of linear and *generalized semi-monadic* TRS [Gyenizse and Vágvölgyi, 1998] and the more general right-linear *finite path overlapping* TRS [Takai et al., 2000].

The TA construction for the rewrite closure for linear and right-shallow TRS is quite similar to the ground TRS case: the TA recognizing the initial language L_{in} is completed by adding new transitions in order to simulate rule applications of $\mathcal{R}$. In a completion step, for each rule $\ell \to f(r_1, \dots r_n)$ in $\mathcal{R}$, ℓ may be not ground. Then, for each substitution σ form $vars(\ell)$ into the state set of the automaton, such that $\sigma(\ell)$ rewrites into a state q using the current transitions rules of the TA, we add the transition $f(q_1, \dots, q_n) \to q$, where for all $i \leq n$, $q_i = q_{r_i}$ if r_i is not a variable (in this case it is a ground term by assumption), and $q_i = \sigma(r_i)$ otherwise. Like for ground TRS, a TA recognizing the rewrite closure is obtained in polynomial time.

The construction for the right-linear and right-shallow TRS [Nagaya and Toyama, 2002] is more involved because these TRS are not left-linear. The trick for the construction is to start with a subset construction, determinizing the initial automaton $\mathcal{A}_{\mathsf{in}}$, and then work on state sets, in order to preserve determinism at each completion step.

The conditions of right-linearity and right-shallowness are not easy to relax. Regarding the right-shallowness, it holds that for instance, with rewrite rules whose left and right hand-side have height at most two, it is possible simulate Turing machine computations, even in the case of words (when all the symbols in the signature are unary or constant). The linear *layered-transducing* TRS, form a particular class of TRS with rules with *lhs* and *rhs* of height more than two and preserving regularity [Seki et al., 2002]. The rules of these TRS have a form similar to those in Figure 7, with a separation of the function symbols in two categories: the states and the others (like in the transitions of bottom-up tree transducers).

Regarding right-linearity, let us consider the following flat and left-linear TRS: $\mathcal{R} = \{f(x) \to g(x, x)\}$, and let L_{in} contain all the terms of the form $f(\dots f(c))$. The intersection of $post^*_{\mathcal{R}}(L_{\mathsf{in}})$ with the regular tree language $\mathcal{T}\big(\{g, c\}\big)$ is the set of balanced binary trees of $\mathcal{T}\big(\{g, c\}\big)$. This set is not regular.

Decision of Properties of Term Rewriting Systems

As mentioned above, a result of effective regularity preservation for a class of TRS ensures the decidability for these TRS of the following property called model checking.

Problem of Model Checking:

Given two TA $\mathcal{A}_{\mathsf{in}}$ and $\mathcal{A}_{\mathsf{err}}$ and a TRS $\mathcal{R}$, decide whether $post^*_{\mathcal{R}}\big(\mathcal{L}(\mathcal{A}_{\mathsf{in}})\big) \cap \mathcal{L}(\mathcal{A}_{\mathsf{err}}) = \emptyset$.

We also noticed that non-reachability is a particular case of model checking, hence its decidability follows from effective regularity preservation.

Problem of Reachability:

Given two terms $s, t \in \mathcal{T}(\Sigma)$ and a TRS $\mathcal{R}$, decide whether $s \xrightarrow[\mathcal{R}]{*} t$.

A consequence of the result of [Nagaya and Toyama, 2002] is the decidability of reachability for right-linear and right-shallow TRS. This also holds for the larger class of finite path overlapping TRS of [Takai et al., 2000]. The decidability of *local confluence* is also a consequence of the effective preserving regularity [Gyenizse and Vágvölgyi, 1998] as well as the decidability of the following property [Gilleron and Tison, 1995].

Problem of Joinability:

Given two terms $s, t \in \mathcal{T}(\Sigma)$ and a TRS $\mathcal{R}$, decide whether there exists a term $u \in \mathcal{T}(\Sigma)$ such that $s \xrightarrow[\mathcal{R}]{*} u \xleftarrow[\mathcal{R}]{*} t$.

It has been conjectured that reachability was decidable for shallow TRS (dropping the right-linearity restriction). A clue for this was in particular that the word problem is decidable for shallow equational theories [Comon et al., 1994]. However, we have shown that it is not the case for TRS.

Theorem 33 [Jacquemard, 2003]

Reachability and joinability are undecidable for flat TRS.

The proof of undecidability of reachability is a somewhat involved reduction of the Post correspondence problem, using coloring techniques of an older proof that we had written with Harald Ganzinger and Margus Veanes for the undecidability of rigid reachability [Ganzinger et al., 1998, 2000], though this latter result could not be reused directly in this context. The TRS for the reduction contains only one non-linear rule of the form $f(\overline{x}) \to g(\overline{x})$ where $\overline{x}$ is a sequence of variables with some repetitions (hence this rule is actually non-left-linear and non-right linear). All the other rules are linear, and their purpose is renaming ($a(x) \to b(x)$) and projection ($a(x) \to x$). The undecidability of joinability follows from a reduction presented

in [Verma et al., 2001]. The above proofs have been simplified later in a joint work with Ichiro Mitsuhashi and Michio Oyamaguchi [Mitsuhashi et al., 2006] and also in [Godoy and Hernandez, 2009]. A light modification of the reduction also permits us to show the undecidability of confluence.

Theorem 34 [Jacquemard, 2003; Mitsuhashi et al., 2006]

Confluence is undecidable for flat TRS.

On the positive side, confluence is decidable for shallow right-linear TRS [Godoy and Tiwari, 2005], and for right-(ground or variable) TRS [Godoy and Tiwari, 2004].

We have already highlighted that term rewriting systems can be seen as a computational model, where the evaluation consists in the application of rewrite rules, starting from a given term, until a normal form is reached, which is considered the result of the computation – see for instance the example with the functions app and rec on lists in Section 1. In this context, an important property of TRS is whether this evaluation process is functional, *i.e.* whether the computation with a TRS is always unique.

Problem of Unique Normalization (UN):

Given a TRS $\mathcal{R}$ over Σ, decide whether for all ground term $t \in \mathcal{T}(\Sigma)$, the set $\mathsf{NF}_{\mathcal{R}}(t)$ has a cardinality at most one.

We have proved results for this problem with Guillem Godoy.

Theorem 35 [Godoy and Jacquemard, 2009]

UN is decidable in PTIME for shallow and linear TRS.

The proof is based on some necessary and sufficient conditions for UN for shallow and linear TRS. The conditions are expressed as properties of some characteristic sets of terms, or more precisely pairs of sets of terms, called *fork of languages*, which, intuitively, characterize pairs of terms $\langle t_1, t_2 \rangle$ which are susceptible to be the first rewrite steps from a common term s towards two distinct normal forms. The properties of forks of languages are shown to be verifiable by reduction to TA decision problems such as emptiness and cardinality. The polynomial time upper bound is achieved through a careful analysis of the size of TAs constructed during the reduction and the complexity of the decision procedures.

This result is very near to the limits of decidability, since unique normalization is known to be undecidable for very restricted classes like right-ground TRS [Verma, 2008], flat TRS [Godoy and Hernandez, 2009], and also linear and right-flat TRS [Godoy and Tison, 2007]. We have proved another undecidability result for UN.

Theorem 36 [Godoy and Jacquemard, 2009]

UN is undecidable for flat and right-linear TRS.

This result is in contrast with the fact that many other natural properties of TRS like reachability, termination, confluence, weak normalization, etc. are decidable for this class of TRS.

A tree automata based method for proving termination of left-linear term rewriting systems on a given regular language of terms is proposed in [Geser et al., 2007]. Note that termination has been shown decidable for right-shallow right-linear TRS [Godoy et al., 2007] and other variants of syntactic restrictions based on the form of the dependency pairs obtained from a TRS [Wang and Sakai, 2006]. Termination is undecidable for flat TRS [Godoy et al., 2007].

Perspective: Shallow and non-Collapsing TRS

The above proof of undecidability of reachability for flat TRS uses non-linear variables in *lhs* and *rhs* of the same rewrite rule, and collapsing (projection) rules of the form $a(x) \to x$. We are currently studying with Masahiko Sakai the importance of these 2 conditions for the undecidability of reachability.

The result [Nagaya and Toyama, 2002] of preservation of regularity for right-shallow and right-linear TRS implies the decidability of reachability for this class of TRS but also for the symmetric class of left-shallow, left-linear and non-collapsing TRS. The non collapsing condition stems from the fact that *lhs* of rewrite rules cannot be variables.

The question of the decidability of reachability is open for for shallow and left linear TRS (including collapsing rules). We think that tree automata technique can help for this case, for instance, the $post^*$ of regular tree sets could be recognized by a variant of TAB (tree automata with brother constraints). The case of shallow (without restriction on linearity) and non-collapsing TRS is also open, and seems difficult.

1.3 Context-Free Term Rewriting Systems

It has been observed, see *e.g.* [Hofbauer and Waldmann, 2004], that in several cases, one class of word rewrite system preserves regularity and its symmetric class preserves CF languages. This is the case for instance of monadic semi-Thue systems, whose rules are length reducing and with a *rhs* of length at most one.

The conditions of flatness (or shallowness) and linearity for *rhs* of rewrite rules ensure the preservation of regularity, even for rules with arbitrary *lhs* of rewrite rules [Nagaya and Toyama, 2002]. The symmetric rules, with flat and linear *lhs*, correspond exactly to productions of context-free tree grammars (presented in Section 2.5) and it can be observed that they preserve context-free tree languages.

Let us call a TRS over Σ *context-free* (CF) if its rules all have the form

$$f(x_1, \ldots, x_n) \to r$$

where $r \in \mathcal{T}\big(\Sigma, \{x_1, \ldots, x_n\}\big)$, and $x_1, \ldots, x_n$ are distinct variables. Recall that when $r = x_i$ for some $i \leq n$, then the rule is called collapsing.

Theorem 37

For all CF tree grammar $\mathcal{G}$ and CF TRS $\mathcal{R}$, a CF tree grammar $\mathcal{G}'$ such that $\mathcal{L}(\mathcal{G}') = post^*_{\mathcal{R}}(L)$ can be constructed in PTIME.

It follows that model-checking and reachability are decidable for CF TRS (for model checking, we use the fact that the intersection of a CF tree language and a regular tree language is a CF tree language). In Section 3.2, we propose the definition of a class of *context-sensitive* tree languages, which is closed by rewriting with linear and size non-decreasing rewrite rules.

1.4 Closure of Languages of Tree Automata with Constraints

Results of preservation (of tree languages) under rewriting are much more difficult to obtain for classes of tree automata with equality and disequality constraints than for standard tree automata. The reason is the difficulty to capture the behavior of the constraints after the application of rewrite rules. To our knowledge there have been only few works on this problem. We present in the next paragraphs two case studies and one idea to overcome this difficulty.

Closure of TAB

It was already observed in [Jacquemard et al., 1998] (joint work with Christoph Meyer and Christoph Weidenbach) that the class of languages of the TAB of [Bogaert and Tison, 1992] (tree automata with local equality tests between brother positions, see Section I.1.1) is not closed under rewriting with flat TRS.

Theorem 38 [Jacquemard et al., 1998]

There exists a regular tree language L and a flat TRS $\mathcal{R}$ such that the closure $post^*_{\mathcal{R}}(L)$ is not recognizable by a TAB.

Note that Theorem 38 is a consequence of Theorem 33 and the decidability of emptiness for TAB. There are simpler counter-examples than the TRS of Theorem 33 for this fact. Let us come back for instance to the example of Section 1.2, $\mathcal{R} = \{f(x) \to g(x, x)\}$. The set $post^*_{\mathcal{R}}\big(\mathcal{T}(\{f, a\})\big) \cap g(\mathcal{T}(\{g, a\}), \mathcal{T}(\{f, a\}))$ is $\{g(t_1, t_2) \mid t_1 \in \mathcal{T}(\{g, a\}), t_2 \in \mathcal{T}(\{f, a\}), h(t_1) = h(t_2)\}$ and it is not recognizable by a TAB.

The situation is not the same when applying the innermost strategy for rewriting. This issue is presented in Section 3.1. Recall that a tree automata model with brother equality modulo flat equational theories was shown decidable in [Jacquemard et al., 1998] (Theorem 15).

Closure of RTA Languages

In [Jacquemard et al., 2011a], we study with Francis Klay and Camille Vacher the closure of languages of tree automata with global constraints. We show that the closure of a RTA language under rewriting is generally not a RTA language, even for a very restrictive class of TRS.

Theorem 39 [Jacquemard et al., 2011a]

In general $post^*_{\mathcal{R}}(L)$ is not an RTA language when L is an RTA language and $\mathcal{R}$ a linear and collapsing TRS.

Restricting to the terms of the rewrite closure in normal form does not help: the intersection of $post^*_{\mathcal{R}}\big(\mathcal{L}(\mathcal{A})\big)$ with $\mathsf{NF}_{\mathcal{R}}$ is not an RTA language in general, when $\mathcal{A}$ is an RTA and $\mathcal{R}$ a linear and collapsing TRS. This situation is in contrast with TA languages, which are closed under rewriting with such TRS [Salomaa, 1988].

However, the preservation of automata languages is not always necessary for systems verification in these settings. For instance, it could be often sufficient to consider the following particular case of the model checking problem where, instead of a TA A_{err} we have a single term t representing an erroneous configuration.

Problem of Membership Modulo:

Given a TRS $\mathcal{R}$ over Σ, an automaton $\mathcal{A}$ and a ground term $t \in \mathcal{T}(\Sigma)$ decide whether $t \in post^*(\mathcal{L}(\mathcal{A}))$.

This problem is unfortunately undecidable for the above class of TRS.

Theorem 40 [Jacquemard et al., 2011a]

Membership modulo is undecidable for RTA and linear and collapsing TRS.

Invisibly Pushdown Term Rewriting Systems

We show in [Jacquemard et al., 2011a] that the problem of membership modulo becomes decidable with some further syntactic restrictions on the TRS, inspired by the definition of visibly pushdown automata [Alur and Madhusudan, 2004]. We

have mentioned in Section I.2.5 a model of visibly pushdown tree automata defined in [Chabin and Réty, 2007] (called VPTA in [Chabin and Réty, 2007] but which is incomparable in expressiveness with the model called VPTA in Section I.2.5). It is shown in [Chabin and Réty, 2007] that the class of languages of these automata is closed under rewriting with so called linear visibly context-free TRS. We use a similar definition in order to characterize a class of TRS *wrt* which membership modulo is decidable for RTA.

Let us assume a partition of the signature Σ into $\Sigma_c \uplus \Sigma_r \uplus \Sigma_\ell$. A collapsing TRS $\mathcal{R}$ is called *inverse-visibly pushdown* (invisibly pushdown) if for every rule $\ell \to x \in \mathcal{R}$, $d(\ell) \geq 1$, x occurs once in ℓ, and if x occurs at height 1 in ℓ then $\ell \in \mathcal{T}(\Sigma_\ell, \mathcal{X})$, otherwise, $\ell(\varepsilon) \in \Sigma_c$, the symbol immediately above x is in Σ_r and all the other symbols of ℓ are in Σ_ℓ.

Linear and invisibly pushdown TRS can typically specify cryptographic operators with rules like $\mathsf{decrypt}(\mathsf{encrypt}(x, \mathsf{pk}(A)), \mathsf{sk}(A)) \to x$, see also Example 12. We present in [Jacquemard et al., 2011a] an application of RTA to the verification of problems like those mentioned in Section I.2.7 (see also the discussion in Section I.3.2).

Theorem 41 [Jacquemard et al., 2011a]

Membership modulo is decidable for RTA and linear and invisibly pushdown TRS.

The decision algorithm is quite involved. It is based on the construction of a visibly pushdown automaton recognizing the language of ancestors of t *wrt* $\mathcal{R}$ ($pre^*_{\mathcal{R}}(\{t\})$) that belong to $\mathcal{L}(\mathcal{A})$.

Perspective: Saturation of Clausal Representation of RTA

The difficulty of the problem of membership modulo stems from the fact that a RTA defines syntactic equalities constraints, and we want to consider the closure of languages under rewriting with a TRS.

In section I.3.2, we mentioned the opportunity to study models of tree automata with parameters (rigid tree automata) defined as combinations of clauses (rta) and a generalization of (eq) (Section I2.2) with parameters. This would be a model of RTA modulo equational theory, where equality is also considered modulo the same theory, an leaves hope for simpler techniques for proving membership modulo.

In particular, a termination results for saturation of sets of such clauses under a *e.g.* a paramodulation calculus would provide an interesting alternative to the decision algorithms presented in [Jacquemard et al., 2009, 2011a].

2 Unranked Tree Rewriting

The examples of applications of regular model checking techniques presented in Section 1 were restricted to ranked terms (standard term rewriting systems and ranked

tree automata). In some cases though, the tree structure to be manipulated consist in unranked trees. For instance, it is convenient to consider unranked tree when dealing with signatures containing binary function symbols modulo A and modulo AC, like respectively the sequential composition $\cdot$ and the parallel composition $\|$ used in Section 1 for the analysis of imperative programs with recursive procedure calls and thread creation. It is common to consider the associative symbols as *variadic*, *i.e.* to assume that these symbols can have an arbitrary number of children in trees. In [Bouajjani and Touili, 2005], a model of tree automata computing on unranked trees, related to other models [Colcombet, 2002; Seidl et al., 2003; Dal Zilio and Lugiez, 2006] is used in a regular model checking procedure. Labeled unranked trees are also commonly used as an abstract model of XML documents, and we discuss the role of unranked tree languages in this context in the next paragraph.

Typechecking Unranked Tree Transformations

A typical case where the data manipulated by programs is represented by unranked trees is the analysis XML transformations. We have seen in Section I.3.6 that XML documents are often constrained by a type definition expressed as an automaton computing on unranked trees (we will give the precise definition classes of tree automata below in Section 2.1). A central problem in this context is *static typechecking* [Milo et al., 2003], which amounts to verifying at compile time that every output XML document which is the result of a specified query or transformation applied to an input document with a valid input type has a valid output type.

Problem of Typechecking:

Given two tree automata $\mathcal{A}_{\mathsf{in}}$ and $\mathcal{A}_{\mathsf{out}}$ and a tree transformation T, decide whether $T\big(\mathcal{L}(\mathcal{A}_{\mathsf{in}})\big) \subseteq \mathcal{L}(\mathcal{A}_{\mathsf{out}})$.

It is equivalent to check whether $\mathcal{L}(\mathcal{A}_{\mathsf{in}}) \cap pre^*_{\mathcal{R}}\big(\overline{\mathcal{L}(\mathcal{A}_{\mathsf{out}})}\big) = \emptyset$, where $\overline{\mathcal{L}(\mathcal{A}_{\mathsf{out}})}$ is the complement of $\mathcal{L}(\mathcal{A}_{\mathsf{out}})$. Typechecking is therefore closely related to the problem of model checking of Section 1. A standard approach to XML typechecking is forward (resp. backward) *type inference*, that is, the computation of an output (resp. input) XML type (as a tree automaton) given an input (resp. output) type and a tree transformation. Then the typechecking itself can be reduced to the verification of set operations on the computed input or output type, see [Milo et al., 2003] for an example of backward type inference procedure.

2.1 Hedge Automata and CF-Hedge Automata

We consider a finite alphabet Σ, *i.e.* a set of symbols *without* arity. The set $\mathcal{H}(\Sigma, \mathcal{X})$ of *hedges* over Σ and $\mathcal{X}$ is the set of finite (possibly empty) sequences of unranked ordered trees and the set of *unranked ordered trees* over Σ and $\mathcal{X}$ is $\mathcal{O}(\Sigma, \mathcal{X}) := \mathcal{X} \cup \big\{a(h) \;\big|\; a \in \Sigma, h \in \mathcal{H}(\Sigma, \mathcal{X})\big\}$. The empty sequence is denoted $()$ and when h

is empty, the tree $a(h)$ will be simply denoted by a. A root of a hedge $(t_1 \dots t_n)$ is a root position of one of $t_1, ..., t_n$. We will sometimes consider a tree as a hedge of length one, *i.e.* consider that $\mathcal{O}(\Sigma, \mathcal{X}) \subset \mathcal{H}(\Sigma, \mathcal{X})$. The sets of ground trees (trees without variables) and ground hedges are respectively denoted $\mathcal{O}(\Sigma)$ and $\mathcal{H}(\Sigma)$.

Hedge Automata and CF-Hedge Automata

The two dimensions, horizontal and vertical, for the navigation in the above unranked ordered trees are captured by the following definitions of unranked tree automata. The hedge automata [Murata, 2000] captures the expressive strength of almost all popular type formalisms for XML [Murata et al., 2000]. We consider also a second class, perhaps lesser known, the context-free hedge automata, introduced in [Ohsaki et al., 2003]. The context-free hedge automata are strictly more expressive than the hedge automata and we shall see that they are of interest for computing certain rewrite closures.

A *hedge automaton* (HA), resp. *context-free hedge automaton* (CF-HA), is a tuple $\mathcal{A} = (\Sigma, Q, F, \Delta)$ where Σ is a finite alphabet, Q is a finite set of states disjoint from Σ, $F \subseteq Q$ is a set of final states, and Δ is a set of transitions of the form $a(L) \to q$ where $a \in \Sigma$, $q \in Q$ and $L \subseteq Q^*$ is a regular word language, resp. a context-free word language. The languages L in the transitions can be presented in several ways (finite (pushdown) automata, grammars, alternating automata, regular expression...). Unless otherwise stated, we assume that L is presented by finite automata for HA and context-free grammars for CF-HA. The move relation between ground hedges $h, h' \in \mathcal{H}(\Sigma \cup Q)$ is defined as follows: $h \xrightarrow[\mathcal{A}]{} h'$ iff there exists a context $C \in \mathcal{H}(\Sigma, \{x\})$ and a transition $a(L) \to q \in \Delta$ such that $h = C[a(q_1 \dots q_n)]$, with $q_1 \dots q_n \in L$ and $h' = C[q]$. The relation $\xrightarrow[\mathcal{A}]{*}$ is the transitive closure of $\xrightarrow[\mathcal{A}]{}$.

The language of a HA or CF-HA $\mathcal{A}$ over Σ in one of its states q, denoted by $\mathcal{L}(\mathcal{A}, q)$ and also called the set of hedges of type q, is the set of ground hedges $h \in \mathcal{H}(\Sigma)$ such that $h \xrightarrow[\mathcal{A}]{*} q$. Note that with the above definitions, all hedges in $\mathcal{L}(\mathcal{A}, q)$ are actually trees of $\mathcal{O}(\Sigma)$. The language of $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$, is the union $\bigcup_{q \in F} \mathcal{L}(\mathcal{A}, q)$.

The problems of membership and emptiness are decidable in PTIME for both classes of HA [Murata, 2000; Comon et al., 2007] and CF-HA [Ohsaki et al., 2003].

The HA languages are effectively closed under all Boolean operations (union, intersection, complementation), with PTIME (resp. EXPTIME) constructions of automata of polynomial (resp. exponential) sizes for the closures under union and intersection (resp. complement). The CF-HA are not closed under intersection and complementation, this is a consequence of the same property of CF word languages. The intersection of a CF-HA language and a HA language is a CF-HA language (with a PTIME automata construction by Cartesian product).

Epsilon and Collapsing Transitions

We study with Michael Rusinowitch, in [Jacquemard and Rusinowitch, 2008a, 2010] the extension of HA and CF-HA with so called *collapsing transitions* which are special

transitions of the form $L \to q$ where $L \subseteq Q^*$ is a context-free language and q is a state. The move relation for the extended set of transitions generalizes the above definition with the case $C[q_1 \dots q_n] \xrightarrow[\mathcal{A}]{} C[q]$ if there exists a collapsing transition $L \to q$ of $\mathcal{A}$ and $q_1 \dots q_n \in L$. The definition of the languages of HA and CF-HA is extended to automata with collapsing transitions accordingly. Note that collapsing transitions can reduce a ground hedge of length more than one into a single state. Hence, if $\mathcal{A}$ contains collapsing transitions, then the languages $\mathcal{L}(\mathcal{A}, q)$ may contain some hedges of $\mathcal{H}(\Sigma) \setminus \mathcal{O}(\Sigma)$.

A collapsing transition with a singleton language L containing a length one word (*i.e.* transitions of the form $\{q'\} \to q$, where q' and q are states) correspond to *ε-transitions* for tree automata, and we use the same name here. The ε-transitions do not increase the expressiveness HA or CF-HA (see [Comon et al., 2007] for HA and the proof for CF-HA is similar), but collapsing transitions strictly extend HA in expressiveness, and even collapsing transitions of the form $L \to q$ where L is finite (hence regular).

Example 16 *The extended HA*

$$\mathcal{A} = \big(\{q, q_a, q_b, q_\mathsf{f}\}, \{g, a, b\}, \{q_\mathsf{f}\}, \{a \to q_a, b \to q_b, g(q) \to q_\mathsf{f}, q_a\, q\, q_b \to q\}\big)$$

recognizes $\{g(a^n b^n) \mid n \geq 1\}$ *which is not a HA language.* ◇

	ε-transitions	collapsing-transitions
HA	HA	CF-HA
CF-HA	CF-HA	CF-HA

Table 4: Extensions of HA and CF-HA with ε- and collapsing-transitions.

We show with Michael Rusinowitch that collapsing transitions can be eliminated from CF-HA, when restricting to the recognition of trees (and not hedges).

Theorem 42 [Jacquemard and Rusinowitch, 2008a]

Every CF-HA $\mathcal{A}$ extended with collapsing transitions can be transformed in polynomial time into a CF-HA $\mathcal{A}'$ without collapsing transitions such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A}) \cap \mathcal{O}(\Sigma)$.

Relation between HA and Ranked Term Languages

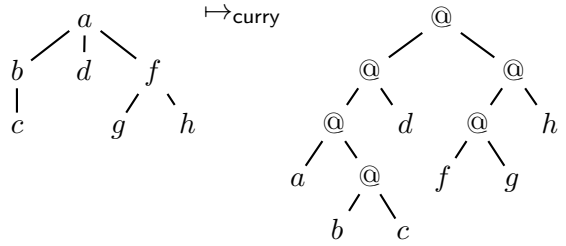
The models of tree automata of Part I compute on ranked terms. Several encodings of unranked ordered trees into ranked terms have been proposed, like the *first-child-next-sibling* encoding [Koch, 2003]. The following transformation [Carme et al., 2004]

$\mathsf{curry} : \mathcal{O}(\Sigma) \to \mathcal{T}(\Sigma_@)$ is particularly well-suited for bottom-up tree recognizers. It is a bijection from unranked ordered trees over a finite alphabet Σ into binary trees over the (ranked) signature $\Sigma_@ := \{a : 0 \mid a \in \Sigma\} \cup \{@ : 2\}$ where $@$ is a new symbol not in Σ.

$$\begin{array}{rcl} \mathsf{curry}(a) & = & a \quad \text{for all } a \in \Sigma \\ \mathsf{curry}\big(a(t_1, \ldots, t_n)\big) & = & @\big(\mathsf{curry}\big(a(t_1, \ldots, t_{n-1})\big), \mathsf{curry}(t_n)\big) \end{array}$$

An example of application of this operator is presented in Figure 9. We extend the

Figure 9 Currying an unranked tree.



application of the operator curry to set of trees by $\mathsf{curry}(L) = \{\mathsf{curry}(t) \mid t \in L\}$.

The HA are expressively equivalent to the TA via this transformation: for all HA $\mathcal{A}$ over Σ, on can construct in polynomial time a TA over $\Sigma_@$ recognizing $\mathsf{curry}\big(\mathcal{L}(\mathcal{A})\big)$. Reciprocally, is it immediate that every TA is a particular case of HA (whose languages in the transitions rules are finite).

Relation between CF-HA Ranked Term Languages

Let us assume a signature Σ, and a distinguished subset Σ_A of binary symbols which follows the associativity axiom.

$$a(x_1, a(x_2, x_3)) = a(a(x_1, x_2), x_3) \tag{A}$$

The function $\mathit{flat} : \mathcal{T}(\Sigma) \to \mathcal{O}(\Sigma)$ associates to every ranked term $t \in \mathcal{T}(\Sigma)$ an unranked ordered tree labeled over Σ representing all the terms equivalent to t modulo (A). The definition of flat in Figure 10 uses intermediate functions $\mathit{hflat}_a : \mathcal{T}(\Sigma)^* \to \mathcal{H}(\Sigma)$, for each $a \in \Sigma_\mathsf{A}$, transforming a sequence of terms into an hedge over Σ. The inverse function $\mathit{flat}^{-1} : \mathcal{O}(\Sigma) \to \mathcal{T}(\Sigma)$ is defined on $\mathcal{O}(\Sigma) \cap \mathit{flat}(\mathcal{T}(\Sigma))$.

The CF-HA are equivalent in expressiveness (via the flattening) to the class of closure of regular ranked term languages modulo (A) (the closure of a language L is denoted $\mathsf{A}(L)$): for all TA $\mathcal{A}$ there exists a CF-HA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathit{flat}\big(\mathsf{A}(\mathcal{L}(\mathcal{A}))\big)$, and for all CF-HA $\mathcal{A}$ there exists a TA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathit{flat}^{-1}\big(\mathcal{L}(\mathcal{A})\big)$. Note that it can happen that the closure modulo (A) of a regular ranked term language is not regular. The above notion of closure of regular languages is captured by the model of *equational tree automata* for associative theories [Ohsaki, 2001].

Figure 10 Definitions of flattening and unflattening operators ($g \in \Sigma_n \setminus \{a\}$).

$$
\begin{aligned}
\mathit{flat}\big(g(t_1,\dots,t_n)\big) &= g(\mathit{flat}(t_1)\dots\mathit{flat}(t_n)) \quad \text{for all } g \in \Sigma \setminus \Sigma_{\mathsf{A}} \\
\mathit{flat}\big(a(t_1,t_2)\big) &= a(\mathit{hflat}_a(t_1\,t_2)) \quad \text{for all } a \in \Sigma_{\mathsf{A}} \\
\mathit{hflat}_a\big(g(s_1,\dots,s_n)\,t_2\dots t_m\big) &= \mathit{flat}\big(g(s_1,\dots,s_n)\big)\,\mathit{hflat}_a(t_2\dots t_m) \\
&\qquad \text{for all } g \in \Sigma \setminus \{a\} \\
\mathit{hflat}_a\big(a(s_1,s_2)\,t_2\dots t_m\big) &= \mathit{hflat}_a(s_1 s_2\,t_2\dots t_m) \\
\mathit{flat}^{-1}\big(g(t_1\dots t_n)\big) &= g(\mathit{flat}^{-1}(t_1),\dots,\mathit{flat}^{-1}(t_n)) \text{ for all } g \in \Sigma \setminus \Sigma_{\mathsf{A}} \\
\mathit{flat}^{-1}\big(a(t_1\dots t_m)\big) &= a(\mathit{flat}^{-1}(t_1), a(\mathit{flat}^{-1}(t_2),\dots, \\
&\qquad a(\mathit{flat}^{-1}(t_{m-1}), \mathit{flat}^{-1}(t_m)))) \ (m \geq 2), \text{ for all } a \in \Sigma_{\mathsf{A}}
\end{aligned}
$$

2.2 Hedge Rewriting Systems

Inspired by [Touili, 2007], we have studied in [Jacquemard and Rusinowitch, 2008a], with Michael Rusinowitch, a non-standard definition of rewriting, extending the classical TRS of [Dershowitz and Jouannaud, 1990] from ranked terms to unranked trees.

Let us extend first the definition of *substitution* σ to mappings from finite subsets of the variable set $\mathcal{X}$ into the hedges of $\mathcal{H}(\Sigma,\mathcal{X})$ (instead of trees in Section 1). An *hedge rewriting system* (HRS) $\mathcal{R}$ over a finite unranked alphabet Σ is a set of *hedge rewrite rules* of the form $\ell \to r$ where $\ell \in \mathcal{H}(\Sigma,\mathcal{X}) \setminus \mathcal{X}$ and $r \in \mathcal{H}(\Sigma,\mathcal{X})$. Unless otherwise stated, we assume the HRS that we consider to be finite. The notions of (left-, right-) linear, ground and collapsing rules is the same for HRS as for TRS. Then, as in Section 1, the rewrite relation $\xrightarrow[\mathcal{R}]{}$ of a HRS $\mathcal{R}$ is defined as the smallest binary relation on $\mathcal{H}(\Sigma,\mathcal{X})$ containing $\mathcal{R}$ and closed by application of substitutions (by hedges) and contexts. In other words, $h \xrightarrow[\mathcal{R}]{} h'$ iff there exists a context C, a rule $\ell \to r$ in $\mathcal{R}$ and a substitution σ such that $h = C[\sigma(\ell)]$ and $h' = C[\sigma(r)]$. The reflexive and transitive closure of $\xrightarrow[\mathcal{R}]{}$ is denoted $\xrightarrow[\mathcal{R}]{*}$.

Example 17 *With $\mathcal{R} = \{g(x) \to x\}$, we have $g(h) \xrightarrow[\mathcal{R}]{} h$ for all $h \in \mathcal{H}(\Sigma,\mathcal{X})$ (the tree is reduced to the hedge h of its arguments). With $\mathcal{R} = \{g(x) \to g(axb)\}$, $g(c) \xrightarrow[\mathcal{R}]{*} g(a^n c b^n)$ for every $n \geq 0$.* ◇

The decision problems of reachability, joinability and model checking are defined for HRS similarly as for TRS (Section 1.2), with ground hedges instead of ground terms and HA instead of TA.

CF and inverse-CF HRS

In [Jacquemard and Rusinowitch, 2008a], we consider particular cases of rewrite rules: a HRS rule $\ell \to r$ is called

context-free if $\ell = a(x)$ with $a \in \Sigma$ and $x \in \mathcal{X}$ (it is not required that $x \in \mathit{vars}(r)$),

inverse context-free if $r \to \ell$ is context-free,

prefix (resp. *postfix*) if $r = g(t_0 \dots t_n\, x)$ (resp. $r = g(x\, t_0 \dots t_n)$) with $x \in vars(\ell)$ and no variable of ℓ occurs in the trees $t_0, \dots, t_n$.

A HRS is said to have one of the above properties if all its rules have this property. It is shown in [Touili, 2007] how to compute the image of a HA language in one step of rewriting by a right-linear HRS, and how to over-approximate of the rewrite closure of a HA for linear HRS. We propose with Michael Rusinowitch the computation of the exact closure for a class of non-linear HRS.

Theorem 43 [Jacquemard and Rusinowitch, 2008a]

For all HA $\mathcal{A}$ and inverse context-free HRS $\mathcal{R}$, a HA $\mathcal{A}'$ can be constructed such that $\mathcal{L}(\mathcal{A}') = post^*_{\mathcal{R}}(\mathcal{L}(\mathcal{A}))$.

This result is a non-trivial generalization of theorems of [Nagaya and Toyama, 2002] and [Touili, 2007], with proof techniques extending both these former constructions. On one side we generalize [Nagaya and Toyama, 2002] to unranked tree languages. In particular, we start (roughly) by determinizing $\mathcal{A}$ and then complete the HA obtained with new transition rules, according to the rules of $\mathcal{R}$, with the invariant that at each completion step, the HA obtained is deterministic. On the other side, the completion steps follow some principles of the construction of [Touili, 2007], with new constructions and new conditions in order to ensure termination (the construction of [Touili, 2007] is limited to one rewrite step).

As a consequence of Theorem 43, reachability, joinability and model-checking are decidable for inverse context-free HRS.

Relaxing the above assumption on $\mathcal{R}$ in Theorem 43 invalidate the result. We show in particular in [Jacquemard and Rusinowitch, 2008a] that $post^*_{\mathcal{R}}(L)$ is not a HA language in general when L is a HA language and $\mathcal{R}$ is

1. a collapsing HRS with rules of the form $a(x) \to x$. Roughly, in this case, the construction of Theorem 43 has to be adapted by adding some collapsing transitions to the HA, but collapsing transitions strictly extend HA, see Example 16.

2. a context-free, linear and flat HRS, even when the rules of $\mathcal{R}$ are all prefix or postfix.

3. a linear and flat HRS whose rules contain at most two variables: It is possible to simulate Turing Machine computations with such rules.

4. a HRS whose *rhs* of rules are ground or of the form $d(xx)$. We reduce in [Jacquemard and Rusinowitch, 2008a] the blank accepting problem for a Turing machine to reachability for a HRS with right-ground (but not left-linear) rules and one rule of the form $d(xx) \to d'(xx)$.

In the above cases 1 and 2, a CF-HA can be constructed for $post^*_{\mathcal{R}}(L)$ (see Theorem 44 below). In contrast to the above results, in the case of ranked terms, collapsing TRS (even not linear nor flat) preserve regularity [Nagaya and Toyama, 2002], and reachability is decidable in PTIME for the TRS of the above type 3. For HRS, the undecidability result for reachability in this latter case 3 even holds with the strong restriction that rewriting is performed only at the root position, like *e.g.* in this recent application [Fagin et al., 2011] to rewriting of the queries of search engines.

We call a HRS $\mathcal{R}$ *shallow-context-free* if it is context-free, and moreover, for all rule $a(x) \to r \in \mathcal{R}$, x can occurs in r only at height at most 1. Note that this definition includes the case of collapsing rules $a(x) \to x$.

Theorem 44 [Jacquemard and Rusinowitch, 2008a]

For all CF-HA $\mathcal{A}$ and linear shallow-context-free HRS $\mathcal{R}$, a CF-HA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = post^*_{\mathcal{R}}\big(\mathcal{L}(\mathcal{A})\big)$ can be constructed in PTIME.

The proof works by completing the CF grammars in the transitions rules of $\mathcal{A}$, according to the non-collapsing rules of $\mathcal{R}$, and adding collapsing-transitions for the collapsing rules $a(x) \to x$.

A consequence of Theorem 44 is the decidability in PTIME of reachability and model-checking for linear shallow-context-free HRS.

The restrictions of Theorem 44 cannot be relaxed: $post^*_{\mathcal{R}}(L)$ is not a CF-HA language in general when L is a HA language and $\mathcal{R}$ is

1. a linear context-free HRS. A counter example is $\mathcal{R} = \{f(x) \to g(f(ax))\}$ and $L = \{f(c)\}$. In this case, $post^*_{\mathcal{R}}(L)$ is $\big\{\underbrace{g(g(\ldots g(}_{n} f(a^n c)))) \;\big|\; n \geq 0\big\}$.

2. a shallow-context-free HRS (not linear). A counter example is $\mathcal{R} = \{f(x) \to f(xx)\}$ and $L = \{f(a)\}$. Then $post^*_{\mathcal{R}}(L) = \{f(a^n) \mid n = 2^k, k \geq 0\}$.

The results of Theorems 43 and 44 cannot be combined. In other terms, for some HRS containing both linear inverse context-free and linear shallow-context-free rules, the set of descendants of a HA language is not a HA language, neither a CF-HA language and even not recursive.

Theorem 45 [Jacquemard and Rusinowitch, 2008a]

$post^*_{\mathcal{R}}(L)$ is not recursive in general when L is a HA language and $\mathcal{R}$ is a HRS whose rules are either linear inverse context-free or linear shallow-context-free and contain only one variable.

We propose in [Jacquemard and Rusinowitch, 2008a] a reduction of the Post correspondence problem to reachability for HRS of the above form. Finally, we have seen above that context-free HRS do not preserve HA languages. The symmetric result also holds for inverse-context-free HRS and CF-HA languages.

Theorem 46 [Jacquemard and Rusinowitch, 2008a]

$post^*_{\mathcal{R}}(L)$ is not recursive in general when L is a CF-HA language and $\mathcal{R}$ is an inverse context-free HRS.

Encoding into Term Rewriting

Encodings of unranked ordered trees into binary trees, like the one presented in Section 2.1 permit ones to reuse formalisms which operate on ranked terms for reasoning on unranked tree transformations. This approach has been followed for the study of the typechecking problem for XML transformations using tree transducers defined on ranked trees, see *e.g.*[Maneth et al., 2005].

There is however no natural translation of HRS rules compatible with the usual binary encodings. Consider for instance the following inverse-context-free hedge rewrite rule $f(axc) \to f(x)$. For every $n \geq 0$, $f(ab^nc)$ reduces in one step to $f(b^n)$, however there is no finite term rewrite system that can simulate such reductions in one step (see [Jacquemard and Rusinowitch, 2008b] for more details). It is possible to transform one HRS rule as above into several TRS rules operating on binary encodings as expected, however, the rule rules produced would not be in a form known to preserve regularity (typically, this would require *lhs* and *rhs* of height larger than 2). Therefore, it appeared that working directly on the automata for unranked trees is more convenient in this case.

Term Rewriting Modulo Associativity

The HRS actually correspond exactly to TRS modulo associativity [Dershowitz and Jouannaud, 1990], like CF-HA correspond exactly to the closure of TA languages under associativity, as we have seen in Section 2.1. There have been many studies of term rewriting modulo associativity and commutativity, but significantly less studies of term rewriting modulo associativity (A) alone. The above results can be seen alternatively as decision results for term rewriting modulo A, and results of closure by ranked term rewriting modulo A of regular ranked term languages modulo A.

2.3 Parametrized Hedge Rewriting Systems

We propose with Michael Rusinowitch in [Jacquemard and Rusinowitch, 2010] an extension of HRS where the rewrite rules are parametrized by HA languages. It means in particular that a parametrized rule can represent an infinite number of unparametrized rules.

PHRS

Let $\mathcal{A} = (\Sigma, Q, F, \Delta)$ be a HA. An *hedge rewriting system* over Σ *parametrized* by $\mathcal{A}$ (PHRS) is a finite set, denoted $\mathcal{R}/\mathcal{A}$, of rewrite rules $\ell \to r$ where $\ell \in \mathcal{H}(\Sigma, \mathcal{X}) \setminus \mathcal{X}$ and $r \in \mathcal{H}(\Sigma \uplus Q, \mathcal{X})$ and the symbols of Q can only label leaves of r. In this notation, $\mathcal{A}$ may be omitted when it is clear from context or not necessary. The rewrite relation $\xrightarrow[\mathcal{R}/\mathcal{A}]{}$ associated to a PHRS $\mathcal{R}/\mathcal{A}$ is defined as the rewrite relation $\xrightarrow[\mathcal{R}[\mathcal{A}]]{}$ where the HRS $\mathcal{R}[\mathcal{A}]$ is the (possibly infinite) set of all the hedge rewrite rules obtained from the rules $\ell \to r$ in $\mathcal{R}/\mathcal{A}$ by replacing in r every state $p \in Q$ by a ground unranked tree of $\mathcal{L}(\mathcal{A}, p)$. Note that when there are multiple occurrences of a state p in a rule, each occurrence of p is independently replaced with a tree of $\mathcal{L}(\mathcal{A}, p)$, which can generally be different from one another.

Several examples of parametrized rewrite rules can be found in Figure 11, where parameters range over the states p, p_1,.., p_n of a given HA. These rules represent infinite sets of atomic operations of the XQuery update facility [Robie et al., 2011], and some restrictions or extensions. The application is presented in more details in Section 2.4.

Figure 11 Examples of PHRS Rules (XQuery Update Facility Primitives and Extensions).

$a(x)$	$\to$	$b(x)$	REN				
$a(x)$	$\to$	$a(p\,x)$	$\mathsf{INS}_{\mathsf{first}}$	$a(x)$	$\to$	$p\,a(x)$	$\mathsf{INS}_{\mathsf{before}}$
$a(x)$	$\to$	$a(x\,p)$	$\mathsf{INS}_{\mathsf{last}}$	$a(x)$	$\to$	$a(x)\,p$	$\mathsf{INS}_{\mathsf{after}}$
$a(xy)$	$\to$	$a(x\,p\,y)$	$\mathsf{INS}_{\mathsf{into}}$				
$a(x)$	$\to$	p	RPL_1	$a(x)$	$\to$	$p_1 \ldots p_n$	RPL
$a(x)$	$\to$	$()$	DEL	$a(x)$	$\to$	x	$\mathsf{DEL}_{\mathsf{s}}$

Besides the parameters, there are some other differences with the HRS rules studied in Section 2.2 (Theorems 43 and 44). The rules of type $\mathsf{INS}_{\mathsf{before}}$, $\mathsf{INS}_{\mathsf{after}}$, RPL or DEL are really hedge rewriting rules, in the sense that the *rhs* of these rules are hedges of length strictly larger than one. We will see below that such rules have a great importance *wrt* type inference, because they can cause the closure of HA languages to be outside HA languages. Another (less important) difference is the rules of type $\mathsf{INS}_{\mathsf{into}}$ which contain two variables. The restricted form of $\mathsf{INS}_{\mathsf{into}}$ prevents the simulation of Turing machine mentioned above.

We call UFO the class of PHRS rules presented in Figure 11, and $\mathsf{UFO}_{\mathsf{reg}}$ the subclass of PHRS rules of type REN, $\mathsf{INS}_{\mathsf{first}}$, $\mathsf{INS}_{\mathsf{last}}$, $\mathsf{INS}_{\mathsf{into}}$, RPL_1, or DEL.

Forward and Backward Type Inference for PHRS

In [Jacquemard and Rusinowitch, 2010], we show forward and backward type inference results for arbitrary finite iterations of parametrized hedge rewrite rules of the above types, taken in a given set.

Theorem 47 [Jacquemard and Rusinowitch, 2010]

Let $\mathcal{A}$ be a HA, $\mathcal{R}/\mathcal{A}$ be a PHRS with rules of type $\mathsf{UFO}_{\mathsf{reg}}$, and $\mathcal{B}$ be a HA. Then $post^*_{\mathcal{R}/\mathcal{A}}(\mathcal{L}(\mathcal{B}))$ is the language of an HA of size polynomial and which can be constructed in PTIME in the size of $\mathcal{R}/\mathcal{A}$ and $\mathcal{B}$.

The construction starts with $\mathcal{B}$ and completes it incrementally according to a case analysis of the update rules of $\mathcal{R}/\mathcal{A}$: roughly, for the cases REN, $\mathsf{INS}_{\mathsf{first}}$, $\mathsf{INS}_{\mathsf{last}}$, $\mathsf{INS}_{\mathsf{into}}$, some transitions are added to the the finite automata defining the transitions rules of $\mathcal{B}$, and for the cases RPL_1 and DEL, some ε-transitions are added.

A trick in the construction is to use accelerations for the completion steps corresponding to rules INS_*. For instance, for a rule $a(x) \to a(p\,x)$ of type $\mathsf{INS}_{\mathsf{first}}$, a loop $i_{a,q} \xrightarrow{p} i_{a,q}$ is added, where $i_{a,q}$ is the initial state of the finite automaton recognizing L in the HA transition $a(L) \to q$. This permits us to treat with one new transition an arbitrary number of application of $\mathsf{INS}_{\mathsf{first}}$.

Theorem 47 does not hold for all the rules of UFO: some rules of UFO do not preserve HA languages in general. It is evident for RPL: a rule of the form $a(x) \to p_b p_a p_c$, when the languages of p_a, p_b, p_c are respectively the singleton sets $\{a\}$, $\{b\}$, $\{c\}$ transforms $\{d(a)\}$ into $\{d(b^n\,a\,c^n) \mid n \geq 0\}$. We give in [Jacquemard and Rusinowitch, 2010] other examples for the rules $\mathsf{DEL}_{\mathsf{s}}$ on the one hand and a combination of rules $\mathsf{INS}_{\mathsf{first}}$, $\mathsf{INS}_{\mathsf{last}}$ and REN on the other hand. However, the image of a HA under arbitrary iterations of rewrite rules in the class UFO is a CF-HA.

Theorem 48 [Jacquemard and Rusinowitch, 2010]

Let $\mathcal{A}$ be a HA, $\mathcal{R}/\mathcal{A}$ be a PHRS with rules of type UFO, and $\mathcal{B}$ be a CF-HA. Then $post^*_{\mathcal{R}/\mathcal{A}}(\mathcal{L}(\mathcal{B}))$ is the language of an CF-HA of size polynomial and which can be constructed in PTIME in the size of $\mathcal{R}/\mathcal{A}$ and $\mathcal{B}$.

The principle of the construction is the same as in Theorem 47. The main difference is that we add collapsing-transitions (actually a generalisation which preserves CF-HA languages) for the cases $\mathsf{INS}_{\mathsf{before}}$, $\mathsf{INS}_{\mathsf{after}}$, RPL, DEL_s.

A consequence of Theorem 48 is that reachability and model checking are decidable in PTIME for the PHRS in UFO. Moreover, the problem of typechecking the arbitrary iterations of UFO rules is EXPTIME-complete, and PTIME-complete when the output type is presented by a deterministic and complete HA. The EXPTIME-hardness follows from the fact that the inclusion problem is already EXPTIME-complete for ranked TA (see Part I) and the PTIME-hardness follows from the fact that the inclusion problem is PTIME-hard for deterministic HA (see *e.g.* [Comon et al., 2007]).

We have also studied the backward type inference for UFO rewrite rules.

Theorem 49 [Jacquemard and Rusinowitch, 2010]

Let $\mathcal{A}$ be a HA, $\mathcal{R}/\mathcal{A}$ be a PHRS with rules of type UFO, and $\mathcal{B}$ be a HA. Then $pre^*_{\mathcal{R}/\mathcal{A}}(\mathcal{L}(\mathcal{B}))$ is the language of a HA of size exponential and which can be constructed in EXPTIME in the size of $\mathcal{R}/\mathcal{A}$ and $\mathcal{B}$.

Related Work

Similar results [Löding, 2002] have been obtained for ranked terms, for rewrite rules of the form $L \to R$ which specify the replacement of any element of a regular language L by any element of a regular tree language R. These rules have been shown to preserve regularity. Then [Löding and Spelten, 2007] has extended some of these works to unranked tree rewriting for the case of *subtree and flat prefix rewriting* which is a combination of standard ground tree rewriting and prefix word rewriting on the ordered leaves of subtrees of height 1.

2.4 Application: Analyze of XQuery Updates

The language XQuery has been extended to XQuery Update Facility [Robie et al., 2011] in order to provide convenient means of modifying XML documents or data. The language is a recommendation from W3C and adds imperative operations that permit one *e.g.* to update some parts of a document while leaving the rest unchanged. This includes rename, insert, replace and delete primitive operations at the node level. Compared to other transformation languages (such a XSLT), XQuery Update Facility is considered to offer concise, readable solutions. However, the update operations, can create and delete entire subtrees in documents, modifying a document's structure; hence type inference for the updated documents it not an obvious problem.

In the case of replacement or insertion, the new subtrees in argument (called *content* nodes in [Robie et al., 2011]) are specified by positions within the tree in input (using XPath expressions). A formal model for XQuery Update Facility has been proposed [Benedikt and Cheney, 2009] with languages for update primitives and XQuery Updates and their operational semantics. In this abstract model, the subtrees in arguments are approximated by states of a tree automaton (*type names* of *regular expression types* [Hosoya et al., 2005]), just like the parameters of the PHRS of Section 2.3. The class UFO of PHRS rules, defined in Figure 11, indeed represent the update primitives operations of the XQuery update facility, and some additional operations

REN renames a position: it changes its label from a into b. Such a rule leaves the structure of the tree unchanged.

$\mathsf{INS}_{\mathsf{first}}$ inserts a tree of type p at the first position below a position labeled by a.

$\mathsf{INS}_{\mathsf{last}}$ inserts at the last position and

$\mathsf{INS}_{\mathsf{into}}$ inserts at an arbitrary position below a position labeled by a.

$\mathsf{INS}_{\mathsf{before}}$ (resp. $\mathsf{INS}_{\mathsf{after}}$) inserts a tree of type p at the left (resp. right) sibling position to a position labeled by a.

DEL deletes a whole subtree whose root position is labeled by a.

RPL replaces a subtree by a sequence of trees of respective types $p_1, \ldots, p_n$.

RPL_1 is the particular case of RPL with $n = 1$. Note that DEL is also a special case of RPL, with $n = 0$.

Example 18 *Assume that the data of patients in a hospital is stored in an XML document whose type definition is characterized by a HA $\mathcal{A}$ (generalizing the DTD given in introduction) with the following transition rules*

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{hospital}(\{p_{\mathsf{pa}}, p_{\mathsf{epa}}\}^*) & \to & p_{\mathsf{h}}, & \mathsf{name}(p_c^*) & \to & p_{\mathsf{n}}, \\
\mathsf{patient}(p_{\mathsf{n}}\, p_{\mathsf{s}}) & \to & p_{\mathsf{epa}}, & \mathsf{ssn}(p_c^*) & \to & p_{\mathsf{s}}, \\
\mathsf{patient}(p_{\mathsf{n}}\, p_{\mathsf{s}}\, p_{\mathsf{t}}) & \to & p_{\mathsf{pa}}, & \mathsf{mref}(p_c^*) & \to & p_{\mathsf{med}}, \\
\mathsf{treatment}(p_{\mathsf{med}}\, p_{\mathsf{dia}}\, p_{\mathsf{da}}) & \to & p_{\mathsf{t}}, & \mathsf{diagnosis}(p_c^*) & \to & p_{\mathsf{dia}}, \\
\mathsf{a} \to p_c, \mathsf{b} \to p_c, \mathsf{c} \to p_c \ldots & & & \mathsf{date}(p_c^*) & \to & p_{\mathsf{da}}
\end{array}$$

The state p_{h} is final (it corresponds to the entry point *of a DTD). A* DEL *rule* $\mathsf{patient}(x) \to ()$ *will delete a* patient *and a* $\mathsf{INS}_{\mathsf{last}}$ *rule* $\mathsf{hospital}(x) \to \mathsf{hospital}(x\, p_{\mathsf{pa}})$ *will insert a new* patient*, at the last position below the root position* hospital*. We can ensure that the patient newly added has an empty* treatment *list (to be completed later) using* $\mathsf{hospital}(x) \to \mathsf{hospital}(x\, p_{\mathsf{epa}})$*. A* $\mathsf{INS}_{\mathsf{after}}$ *rule* $\mathsf{name}(x) \to \mathsf{name}(x)\, p_{\mathsf{t}}$ *can be used to insert later a* treatment *next to the patient's* name*.* ◇

We propose also in Figure 11 another update primitive not in [Robie et al., 2011]: $\mathsf{DEL}_{\mathsf{s}}$ deletes a single position p whose children inherit the position. In other words, it replaces the subtree at p with the hedge containing the children of p. This operation can be employed to build security views [Fan et al., 2004] of XML documents under read access control policies, see the discussion on this point in Section 2.5. It can also be useful for updates as well, as shown by the following example.

Example 19 *Assume that some patients of the hospital of Example 18 are grouped in one department like in* $\mathsf{hospital}(\ldots \mathsf{surgery}(p_{\mathsf{pa}}^*) \ldots)$*, and that we want to suppress the department* surgery *while keeping its patients. This can be done with the* $\mathsf{DEL}_{\mathsf{s}}$ *rule* $\mathsf{surgery}(x) \to x$*.* ◇

We have seen that the results presented in Section 2.3 permit ones to solve the problems reachability, model checking and typechecking of transformations defined as the iteration of primitive update operations taken in a given finite PHRS of type UFO.

The results of Theorems 47 and 48 also enable the synthesis of missing input or output types for given PHRS in UFO. Unlike HA, CF-HA are not popular type

schemes, but HA solely are not expressive enough to extend the results of Theorem 48 to the whole class UFO. One may wonder to what extent the CF-HA produced by Theorem 48, is actually an HA. This problem is actually undecidable, since the problem of knowing whether a given CF language is regular is undecidable and every CF language can be described by the closure $post^*_{\mathcal{R}/\mathcal{A}}(\mathcal{L}(\mathcal{B}))$ for some $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{L}(\mathcal{B})$.

Moreover, for a $\mathcal{R}/\mathcal{A} \in$ UFO, if an output type is given, then Theorem 49 provides in EXPTIME an input type, presented as a HA of exponential size.

Perspectives: Scheduling and Selection of Rewrite Positions

The model of [Benedikt and Cheney, 2009] defines particular scheduling for the application of update primitives. This point is discussed below at the end of Section 3.2. The question of the selection of the rewrite positions is also discussed in Section 3.2.

Related Work: Typechecking XML Transformations

Several approaches to typechecking XML transformation languages are based on tree transducers. For instance let us cite macro tree transducers (MTT) [Perst and Seidl, 2004; Maneth et al., 2007], and k-pebble tree transducers (k-PTT) [Milo et al., 2003], a powerful model defined so as to cover relevant fragments of XSLT [Kay, 2003] and other XML transformation languages. Some restrictions on schema languages and on top down tree transducers (on which transformations are based) have also been studied [Martens and Neven, 2004; Engelfriet et al., 2009] in order to obtain PTIME typechecking procedures.

On the one hand, the above class UFO of PHRS rules permit ones to express only a set of atomic update primitives, and its expressiveness cannot be compared to general purpose transformation languages, unlike the above transducer models. Moreover, the form of transducer rules is not captured by the UFO rewrite rules. On the other hand, the main difference between the transducer based and rewrite based approaches *wrt* typechecking is the application strategies. The rewrite based approaches consider arbitrary iterations of rewriting with HRS rules, whereas transducers use specific hedge traversal strategies in order to perform a single top-down or bottom pass on an input tree. This makes the two approaches hardly comparable in expressiveness: each of the primitive update operations of UFO can be solely modeled by a MTT. It is however not clear whether the finite (but unbounded) iterations of updates operations can be easily expressed as a transducer relation.

It is shown in [Milo et al., 2003] that the set of output trees of a k-PTT for a fixed input tree is a HA tree language. In contrast, we have seen that it is not the case for the iteration of UFO operations for which CF-HA are needed. Therefore, such transformation are not expressible as k-PTT. Theorem 48 can be related to the result of [Engelfriet and Vogler, 1985], used in [Maneth et al., 2007] in the context of typechecking XML transformations, which states that the output language of a linear stay MTT can be characterized by a context-free tree grammar (in the case of ranked trees).

2.5 Application: Analyze of XML Access Control Policies

Another important issue for XML data processing is the specification and enforcement of access control policies (ACP). A large amount of work has been devoted to secure XML querying, with a focus on read-only rights in most of the work. The first access control model for XML was proposed by [Damiani et al., 2000] and was extended to secure updates in [C. Lim and Son, 2003]. Static analysis has been applied to XML Access Control in [Murata et al., 2006] to determine if a query expression is guaranteed not to access to elements that are forbidden by the policy. Some work have considered update rights for XQuery Update Facility operations [Fundulaki and Maneth, 2007; Bravo et al., 2008]. In [Jacquemard and Rusinowitch, 2010] and an extension currently in submission, we consider with Michael Rusinowitch two approaches for the static analysis of ACP for XML updates.

Rule Based Specification of ACPs

In this approach, following [Fundulaki and Maneth, 2007; Bravo et al., 2008], an ACP for XML updates is defined as a pair of $\langle \mathcal{R}_\mathsf{a}/\mathcal{A}, \mathcal{R}_\mathsf{f}/\mathcal{A} \rangle$ of PHRS of type UFO representing the update operations respectively allowed forbidden to a user. Such an ACP is called *inconsistent* [Fundulaki and Maneth, 2007; Bravo et al., 2008] if some forbidden operation can be simulated through a sequence of allowed operations, *i.e.* if there exists $t, u \in \mathcal{O}(\Sigma)$ such that $t \xrightarrow[\mathcal{R}_\mathsf{f}/\mathcal{A}]{} u$ and $t \xrightarrow[\mathcal{R}_\mathsf{a}/\mathcal{A}]{}^{*} u$. Such situations may lead to serious security breaches which are challenging to detect according to [Fundulaki and Maneth, 2007].

Example 20 *Assume that in the* $\mathsf{hospital}$ *document of Example 18, it is forbidden to rename a* $\mathsf{patient}$*, that is the following update of type* RPL_1 *is forbidden:* $\mathsf{name}(x) \to p_\mathsf{n}$*. If the following updates are allowed:* $\mathsf{patient}(x) \to ()$ *for deleting a* $\mathsf{patient}$*, and* $\mathsf{hospital}(x) \to \mathsf{hospital}(x\, p_\mathsf{pa})$ *to insert a new* $\mathsf{patient}$*, then we have an inconsistency since the effect of the forbidden update can be obtained by a combination of allowed updates.* ◇

Using the results of Section 2.3, we can verify locally the consistency of ACPs, *i.e.* check whether no sequence of allowed updates starting from a given document can achieve an explicitly forbidden update. More precisely, we solve the following problem.

Problem of Local Inconsistency:

given a HA $\mathcal{A}$ over Σ and a tree $t \in \mathcal{O}(\Sigma)$, an ACP $(\mathcal{R}_\mathsf{a}/\mathcal{A}, \mathcal{R}_\mathsf{f}/\mathcal{A})$ is locally inconsistent if there exists $u \in \mathcal{O}(\Sigma)$ such that $t \xrightarrow[\mathcal{R}_\mathsf{f}/\mathcal{A}]{} u$ and $t \xrightarrow[\mathcal{R}_\mathsf{a}/\mathcal{A}]{}^{*} u$?

Theorem 50 [Jacquemard and Rusinowitch, 2010]

Local inconsistency is decidable in PTIME for UFO based ACPs.

It is shown in [Fundulaki and Maneth, 2007] that inconsistency is undecidable for an ACP defined by a pair of rewrite systems $(\mathcal{R}_\mathsf{a}, \mathcal{R}_\mathsf{f})$ of a kind strictly more general than the above PHRS (roughly, they extend the PHRS with the possibility to select the rewrite positions by XPath expressions, see also Section 3.2). Moreover, for such rewrite systems, the problem whether a given unranked tree t can be obtained from a given tree s using instances of rules of $\mathcal{R}_\mathsf{a}$ which are not in $\mathcal{R}_\mathsf{f}$ is also undecidable [Moore, 2011]. Therefore local consistency is undecidable as well in this case. A decidable fragment is also presented in [Moore, 2011]. It is an open question whether inconsistency is decidable or not for PHRS of type $\mathsf{UFO}_\mathsf{reg}$ or UFO.

DTD Based Specification of ACPs

A second approach is proposed in [Fundulaki and Maneth, 2007] for the definition of ACP for XML updates in presence of a DTD $\mathcal{D}$. The idea, following the principle of DTD-based ACPs [Fan et al., 2004], is to add to $\mathcal{D}$ some security annotations specifying the authorizations for the update operations for XML documents valid for $\mathcal{D}$.

The definition of DTD-based ACPs is proposed in [Fan et al., 2004] with a focus on read access control. This paper also introduces the fundamental notion of *security view*: a view of an XML document exposes all and only the data elements and structure accessible to a given user according to an ACP. Assume for instance that in a document of the form

$$t = \mathsf{hospital}(p_1 \dots p_i\, \mathsf{surgery}(p_{i+1} \dots p_j)\, p_{j+1} \dots p_k)$$

like in Example 19, where $p_1, \dots, p_k$ are trees representing patients (of type p_pa), the position labeled by $\mathsf{surgery}$ is not accessible for reading by some user Alice, while all the patients' data is accessible for reading (including the data of $p_{i+1} \dots p_j$ below $\mathsf{surgery}$). One possibility to represent this restriction is to offer Alice the following *view* of the document t:

$$\mathsf{hospital}(p_1 \dots p_i \perp(p_{i+1} \dots p_j)\, p_{j+1} \dots p_k)$$

where the label $\mathsf{surgery}$ is hidden to her. However, with this view Alice has still the information that patients $p_{i+1} \dots p_j$ belong to a special category, and this *structural* information can be considered as a leak. A solution to avoid this leak is to remove the single position labeled $\mathsf{surgery}$ while keeping its children, using the rule DEL_s described in the previous section. This returns the following tree.

$$v = \mathsf{hospital}(p_1 \dots p_i\, p_{i+1} \dots p_j\, p_{j+1} \dots p_k)$$

We have observed in Section 2.3 that this kind of rule can transform a HA language into a CF-HA languages. This shows the relevance of the above notions of PHRS rules and CF-HA for building security views of XML documents under read access control policies.

The formalism of [Fan et al., 2004; Fundulaki and Maneth, 2007] for DTD based specification of XML ACPs imposes the condition that every document t to which we want to apply an update operation (under the given ACP) must be valid for the DTD $\mathcal{D}$. In our rewrite-based formalism, the latter condition may be expressed by adding global constraints to the PHRS rules of Section 2.3. These global constraints restrict the rewrite relation to trees in a given HA language. Given a HA $\mathcal{A}$, a *hedge rewriting system* over Σ, *parametrized* by $\mathcal{A}$ and *with global constraints* (PGHRS) is given by a PHRS, denoted $\mathcal{R}/\mathcal{A}$, and $L \subseteq \mathcal{O}(\Sigma)$ an HA language. We say that L is the constraint of R. The rewrite relation generated by the PGHRS is defined as the restriction of the relation defined in Section 2.3 to ground unranked tree such that for the application of a rule $\ell \to r \in \mathcal{R}/\mathcal{A}$ to a tree t, we require that $t \in L$. Unfortunately, adding such constraints to parametrized rewrite rules of type REN or RPL makes the reachability undecidable even in the restricted case of non recursive DTD's. We recall that a DTD over Σ is function $\mathcal{D}$ that maps Σ to regular expressions over Σ. The dependency graph of a DTD $\mathcal{D}$ is a directed graph on the set of vertices Σ such that the set of edges contains all (a, b) such that b occurs in the regular expression $\mathcal{D}(a)$. A DTD is *non recursive* if this graph is acyclic.

Theorem 51 [Jacquemard and Rusinowitch, 2010]

Reachability is undecidable for PGHRS with rules of type REN or RPL and and constraint given by a non recursive DTD.

It follows that local inconsistency is undecidable for such PGHRS. It is shown in the extended version of [Jacquemard and Rusinowitch, 2010] that the above result also holds for PGHRS whose rules are ground (without variables nor parameters). Hence the above result can be contrasted with the decidability of reachability for ground term rewriting [Gilleron, 1991].

In [Abiteboul et al., 2009] the authors study, in the context and unranked unordered trees, the more general problem of *satisfiability* for active XML documents, which is the existence of an update of a given document returning a particular result for a given query. This property is shown decidable for insertions constrained by an unordered DTD, but undecidable when they are constrained by an unordered HA.

2.6 Unranked Unordered Tree Rewriting Systems

Several models of automata have been defined to compute on unranked unordered trees (which correspond to terms with symbols modulo commutativity and associativity) [Lugiez, 2005; Seidl et al., 2003] or on mixed unranked trees, containing symbols whose children are ordered and symbols whose children are unordered (which

correspond to terms with symbols modulo associativity and symbols modulo commutativity and associativity) [Colcombet, 2002; Seidl et al., 2003; Bouajjani and Touili, 2005; Dal Zilio and Lugiez, 2006]. Some of the above results on the rewrite closure of unranked tree automata languages can be extended to such automata.

The set of *unranked unordered trees* over a finite alphabet Σ and set of variables $\mathcal{X}$ is $\mathcal{U}(\Sigma,\mathcal{X}) := \mathcal{X} \cup \{a(m) \mid a \in \Sigma, m \in \mathcal{M}(\Sigma,\mathcal{X})\}$, where $\mathcal{M}(\Sigma,\mathcal{X})$ is the class of multisets of unranked unordered trees. The subsets without variables are denoted respectively $\mathcal{U}(\Sigma)$ and $\mathcal{M}(\Sigma)$.

Presburger Tree Automata

We give a definition of tree automata computing on unranked unordered trees from [Seidl et al., 2003]. A *Presburger tree automaton* (PTA) is a tuple $\mathcal{A} = (\Sigma, Q, F, \Delta)$ where Σ is a finite alphabet, $Q = \{q_1, \ldots, q_p\}$ is a finite set of states disjoint from Σ, $F \subseteq Q$ is the subset of final states and Δ is a set of transition rules of the form: $a(\phi) \to q$ with $a \in \Sigma$, $q \in Q$, $\phi = \phi(x_1, \ldots, x_p)$ is a Presburger formula (formula of first-order logic with equality, addition of natural numbers, zero and successor) with exactly one free variable for each state of Q.
The move relation between ground trees $t, t' \in \mathcal{U}(\Sigma \cup Q)$ is defined as follows: $t \xrightarrow[\mathcal{A}]{} t'$ iff there exists a context $C \in \mathcal{U}(\Sigma, \{x\})$ and a transition $a\big(\phi(x_1, \ldots, x_p)\big) \to q \in \Delta$ such that $a \in \Sigma$, $t = C[a(w)]$, with $w = q_{i_1} \ldots q_{i_n}$ for some integers $1 \leq i_1, \ldots, i_n \leq p$, $\mathit{Parikh}(w) \models \phi(x_1, \ldots, x_p)$ and $t' = C[q]$, where the Parikh projection $\mathit{Parikh}(w)$ is the p-uple $(|w|_{q_1}, \ldots, |w|_{q_p})$, $|w|_{q_j}$ representing the number of occurrences of q_j in w. The relation $\xrightarrow[\mathcal{A}]{*}$ is the transitive closure of $\xrightarrow[\mathcal{A}]{}$, and the language of a PTA $\mathcal{A}$ over Σ in one of its states q, denoted by $\mathcal{L}(\mathcal{A}, q)$ is the set of trees $t \in \mathcal{U}(\Sigma)$ such that $t \xrightarrow[\mathcal{A}]{*} q$. The PTA are equivalent in expressiveness to

- the CF-HA such that the words CF language in the transitions are closed under permutations,
- the class of regular ranked tree (terms) languages modulo (AC) or equivalently the AC-tree automata of [Ohsaki, 2001; Ohsaki and Takai, 2002b]. The equivalence is established with the flattening function of Section 2.1 (Figure 10).

Note that the closure modulo AC of a regular ranked tree language is not always regular.

The class of PTA languages is closed under union, intersection and complementation and and PTA have a decidable emptiness problem [Seidl et al., 2003]. For the emptiness decision, one may assume, according to Parikh's Theorem, that every Presburger formula $\phi(x_1, \ldots, x_p)$ of the MTA has been pre-compiled into a NFA $\mathcal{A}$ such that $\mathit{Parikh}\big(\mathcal{L}(\mathcal{A})\big) = \{(n_1, \ldots, n_p) \models \phi(x_1, \ldots, x_p)\}$. An equivalent extension of MSO, with counting constraints, interpreted over the trees of $\mathcal{U}(\Sigma)$ is proposed in [Seidl et al., 2003].

Unranked Unordered Tree Rewriting Systems

Similarly to HRS, we can define rewriting systems over unranked unordered tree, using substitutions which are mappings from a finite subset of the variable set $\mathcal{X}$ into the multisets of $\mathcal{M}(\Sigma, \mathcal{X})$. A *multiset rewriting system* (MRS) $\mathcal{R}$ over a finite unranked alphabet Σ is a finite set of rewrite rules of the form $\ell \to r$ where $\ell \in \mathcal{M}(\Sigma, \mathcal{X}) \setminus \mathcal{X}$ and $r \in \mathcal{M}(\Sigma, \mathcal{X})$. The rewrite relation $\xrightarrow[\mathcal{R}]{}$ associated to a MRS $\mathcal{R}$ is defined as the smallest binary relation on $\mathcal{M}(\Sigma, \mathcal{X})$ containing $\mathcal{R}$ and closed by application of substitutions (by multisets) and contexts. In other words, $m \xrightarrow[\mathcal{R}]{} m'$, iff there exists a context C, a rule $\ell \to r$ in $\mathcal{R}$ and a substitution σ such that $m = C[\sigma(\ell)]$ and $m' = C[\sigma(r)]$. The reflexive and transitive closure of $\xrightarrow[\mathcal{R}]{}$ is denoted $\xrightarrow[\mathcal{R}]{*}$. The definition of shallow-context-free MRS is the same as for HRS in Section 2.2. In an unpublished work, we show with Michael Rusinowitch that the result of Theorem 44 can be transposed to PTA and MRS.

Theorem 52

For all PTA $\mathcal{A}$ and all linear shallow-context-free MRS $\mathcal{R}$, a PTA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = post^*_{\mathcal{R}}\big(\mathcal{L}(\mathcal{A})\big)$ can be constructed in PTIME.

The construction is very similar as in Theorem 44, with the completion of Presburger formulae (instead of CF grammars) and the addition of collapsing transitions (for PTA, the collapsing transitions are of the form $\phi(x_1, \ldots, x_p) \to q$ and can be eliminated).

The related model of Presburger weighted tree automata [Lugiez, 2009] is used for forward reachability analysis of dynamic networks of pushdown systems, a model for concurrent programs with dynamic thread creation.

3 Rewrite Strategies

Many strategies have been proposed for controlling explicitly the application of term rewriting rules, for instance in the cases when the computation result may be not unique, or from a practical point of view, for termination and efficiency. We describe below some work using tree automata techniques for the analysis of rewrite relations under various strategies, with a focus on the representation by tree automata of the rewrite closures of regular tree languages.

The rewrite closures of tree languages have been extensively studied for plain rewriting (see for instance the results presented in Section 1.2), but much less for rewriting with strategies. We present below some studies of the automata representation of the rewrite closures with various strategies. Different classes of extended tree automata are needed for these studies.

Bottom-up and Bounded Strategies

The *bottom-up* strategy [Durand and Sénizergues, 2007] is defined for linear term rewriting systems and imposes, roughly speaking, that in the rewrite derivations, the rewrite rules are applied from the bottom of the term towards the top. It is shown in [Durand and Sénizergues, 2007] that bottom-up rewriting effectively inverse-preserves recognizability for linear TRS: the backward closure $pre^*_{\mathcal{R}}(L)$ of a regular tree language L is regular.

The bottom-up strategy has been refined into the *bounded* strategy [Durand et al., 2010; Durand and Sylvestre, 2011], which is parametrized by an integer k and such that, roughly, when a rewrite rule is applied, the parts of the substitution located at a height greater than k are not rewritten further. This strategy is shown to inverse-preserves recognizability for linear TRS [Durand et al., 2010] and left-linear TRS [Durand and Sylvestre, 2011].

These strategies can also serve as defining a class of TRS which inverse-preserve regularity: the bounded class (BO) of TRS is the set of left-linear TRS for which every derivation can be replaced by a bottom-up derivation. This approach is an alternative to the syntactical definitions presented *e.g.* in Section 1.2, and the class BO is actually strictly more general than several classes of TRS which were already known to be inverse-recognizability preserving, like the symmetric of left-linear semi-monadic TRS or right-linear finite-path overlapping TRS.

Context-Sensitive Strategy

With the *context-sensitive* strategy [Futatsugi et al., 1985] (see also [Lucas, 2002]), the rewriting positions are restricted according to a mapping μ associating to every symbol of the signature the subset of the indexes of its argument that can be rewritten. More precisely, the positions selected for rewriting in a term $f(t_1, \ldots, t_n)$ are defined recursively as the root position and all the positions selected in every t_i such that $i \in \mu(f)$. A result of preservation of regularity for this rewrite strategy is established in [Kojima and Sakai, 2008] for linear right-shallow TRS. Context-sensitive is related to the strategy presented in Section 3.2.

Call-by-Need strategies

Another related topic is the optimal *call-by-need* rewrite strategies for TRS. In [Comon, 1995] and [Durand and Middeldorp, 2005], it is shown how to select the *needed redexes* (positions at which rewriting must be performed in order to transform a term to its normal form) using monadic second order logic formulae, which corresponds to using the selection automata presented in Section 3.2. Besides the detection of needed redexes, tree automata techniques have also been used in order to characterize the class of TRS on which such strategy is effective. The idea for this purpose is, given a TRS $\mathcal{R}$, to compute the inverse closure of the set of $\mathcal{R}_0$-ground normal forms for a TRS $\mathcal{R}_0$ over-approximating $\mathcal{R}$ (in the sense that the binary relation $\xrightarrow[\mathcal{R}_0]{*}$ is an over-approximation of $\xrightarrow[\mathcal{R}]{*}$) see [Durand and Middeldorp, 2005].

3.1 Innermost Strategies

The *innermost rewrite strategy* corresponds to the *call by value* computations for functional languages, where the arguments must be fully evaluated before the function application. More precisely, a rewrite step $s \xrightarrow[\mathcal{R},p,\sigma]{} t$ is called *innermost* if all proper subterms of $s|_p$ are $\mathcal{R}$-normal forms. In this case, we write $s \xrightarrow[\mathcal{R}]{i} t$, and the transitive and reflexive closure of this relation is denoted by $\xrightarrow[\mathcal{R}]{i*}$, and $post^{i*}_{\mathcal{R}}(L)$ is $\{t \mid \exists s \in L, s \xrightarrow[\mathcal{R}]{i*} t\}$.

Some tree automata constructions are proposed in [Réty and Vuotto, 2005] to express the sets of descendants of regular languages by constructor-based linear rewrite systems, according to several strategies, including innermost, outermost, leftmost and innermost-leftmost, and under various conditions.

In [Gascón et al., 2008], with Adria Gascon and Guillem Godoy, we consider innermost rewriting with shallow TRS, without assumptions on the linearity of rules. In all the known cases of TRS preserving regularity, including those presented in Section 1.2, the condition of right-linearity is necessary. In fact, we have seen in Section 1.2 that a TRS as simple as $\mathcal{R} = \{g(x) \to f(x,x)\}$ does not preserve regularity, because $post^*_{\mathcal{R}}(\mathcal{T}(\{g,a\})) \cap \mathcal{T}(\{f,a\})$ is the set of balanced binary trees of $\mathcal{T}(\{f,a\})$, that we denote $\mathcal{B}_{f,a}$. Note that this set is a TAB language. However, we have also seen (Theorem 38) that the rewrite closure of a regular tree language by a flat TRS is not a TAB language in general. The essential problem relies on the fact that after an application of the rule $g(x) \to f(x,x)$ on a subterm $g(t)$, producing $f(t,t)$, some further applications of this rewrite rule in $f(t,t)$ can modify both occurrences of t in different ways, producing subterms $f(t_1,t_2)$ with $t_1 \neq t_2$ but $h(t_1) = h(t_2)$. The equality constraints of TAB are not expressive enough to capture the relation between t_1 and t_2, which corresponds to the fact that the both are reachable from a common term.

The above problem does not occur with innermost rewriting. In the above example, when we apply the rule $g(x) \to f(x,x)$ with the innermost strategy, the subterm to whom it is applied has to be of the form $g(t)$, for a normal form t. Hence, in the term obtained $f(t,t)$, the two occurrences of t cannot be further modified by rewriting. In fact, the innermost closure $post^{i*}_{\mathcal{R}}(\mathcal{T}(\{g,a\}))$ is $\{g^n(t) \mid t \in \mathcal{B}_{f,a}\}$ and this set is a TAB language. We generalize this result to all shallow TRS with Adria Gascon and Guillem Godoy.

Theorem 53 [Gascón et al., 2008]

For all TA $\mathcal{A}$ and all shallow TRS $\mathcal{R}$, $post^{i*}_{\mathcal{R}}(\mathcal{L}(\mathcal{A}))$ is a TAB language.

Note that in the above example, $post^{i*}_{\mathcal{R}}(T)$ is a TAB language which is not regular. The technique of tree automata completion, which works well with TA and *e.g.* linear and shallow TRS, see Sections 1.1 and 1.2, cannot be generalized to TAB, neither for plain nor innermost rewriting. Indeed, in general, $post^*_{\mathcal{R}}(L)$ and $post^{i*}_{\mathcal{R}}(L)$ are not TAB languages when L is a TAB language and $\mathcal{R}$ is a flat and linear TRS.

Example 21 ([Gascón et al., 2008]) *Consider for instance the non-regular* TAB *language*
$L = \{h(f^n(a), f^n(a)) \mid n \geq 0\}$ *and the linear and flat TRS* $\mathcal{R} = \{f(x) \to g(x)\}$.

It holds that $post^*_{\mathcal{R}}(L) \cap \{h(f^n(0), g^m(0)) \mid m, n \geq 0\} = \{h(f^n(0), g^n(0))\}$, *which is not a* TAB *language. This also holds when we restrict to innermost rewriting.* ◇

The proof of Theorem 53 works in two steps. First, we reduce the problem of representing the reachable terms from a regular set to the reachable terms from a constant. Next, we give a direct construction of a TAB recognizing the reachable terms from a constant. It is based on a representation of the set of reachable terms introduced in [Godoy and Huntingford, 2007] using constrained terms.

A consequence of Theorem 53 is that innermost reachability and joinability are decidable for shallow TRS. This is in contrast with Theorem 33 for plain rewriting. Another consequence is that it is decidable whether $post^{\wedge}_{\mathcal{R}}\big(\mathcal{L}(\mathcal{A})\big)$ is regular given a TA $\mathcal{A}$ and a shallow TRS $\mathcal{R}$, because the regularity of TAB is decidable (see [Bogaert et al., 1999] and Section I.1.4). In contrast the regularity of $post^*_{\mathcal{R}}\big(\mathcal{L}(\mathcal{A})\big)$ is undecidable under the same assumptions [Gascón et al., 2008].

We also show that, like for plain rewriting, innermost rewriting with linear and right-shallow TRS preserves regularity.

Theorem 54 [Gascón et al., 2008]

For all TA $\mathcal{A}$ and all linear and right-shallow TRS $\mathcal{R}$, a TA $\mathcal{A}'$ can be constructed in PTIME such that $\mathcal{L}(\mathcal{A}') = post^{\wedge}_{\mathcal{R}}\big(\mathcal{L}(\mathcal{A})\big)$.

This result has been independently obtained in [Kojima and Sakai, 2008]. In our case, it is proved with a non trivial adaptation of the tree automata completion technique. The cases of plain and innermost rewriting are different in essence to treat, and some subtle differences need to be introduced. We show in particular [Gascón et al., 2008] that even though TA completion permits ones to establish that right-linear and right-flat TRS (*i.e.* when left-hand sides of rules might be not linear) preserve regular languages under plain rewriting, this property is no longer true for under innermost rewriting. A counter example is the following.

Example 22 ([Gascón et al., 2008]) *Let* $L = \{f(f(a, a), c)\}$, *and* $\mathcal{R} = \{f(x, c) \to x, f(g(x), x) \to h(x), h(x) \to h(x), a \to g(a), a \to b\}$. *The intersection* $post^{\wedge}_{\mathcal{R}}(L) \cap \mathcal{T}\big(\{f, g, b\}\big)$ *is the set* $\{f(g^n(b), g^m(b)) \mid n \neq m + 1\}$, *which is not a* TAB *language.*◇

3.2 Context-Controlled Rewriting

With the definition of Section 1, a rewrite rule can be applied at any position in a term, providing that the *lhs* of the rule matches the subterm at this position. For instance, a rule with *lhs* $a(x_1, x_2)$ can be applied at any position labelled by a. For some applications like the analysis of XML document transformations or of access-control policies, presented in Sections 2.4 and 2.5, it is important to be able to express

explicitly some *context conditions* to be checked before applying a rewrite rule. For instance, one may want to rename the label at some position with a rewrite rule $a(x) \to b(x)$ providing that there is no occurrence of b above the position to be rewritten (position labeled with a). Some standard XML transformation languages like XSLT or XQuery update [Robie et al., 2011], use the path specification language XPath or a XQuery expressions in order to define the position where the transformation can be applied.

We studied with Yoshiharu Kojima and Masahiko Sakai [Jacquemard et al., 2011b] a strategy of selection of the rewrite positions based on tree automata, in the context of regular tree model checking. The so called *controlled term rewriting systems* are defined by the combination of term rewriting rules with constraints (called control) specifying the possible rewrite positions.

Selection Automata

In order to define constraints controlling the rewrite rules, we have chosen in [Jacquemard et al., 2011b] a model called selection tree automata (SA), similar to [Frick et al., 2003], which, intuitively, select positions in a term based on the computations of a tree automaton. A *selection automaton* (SA) $\mathcal{A}$ is a tuple $\langle \Sigma, Q, F, S, \Delta \rangle$ where $\langle \Sigma, Q, F, \Delta \rangle$ is a tree automaton denoted $ta(\mathcal{A})$ and S is a set of states of Q called selection states. Given a SA $\mathcal{A}$ and a term $t \in \mathcal{T}(\Sigma)$, the set of positions of t selected by $\mathcal{A}$ is defined as

$$sel(\mathcal{A}, t) = \{p \in \mathcal{P}os(t) \mid \exists r \text{ successful run of } ta(\mathcal{A}) \text{ on } t, r(p) \in S\}.$$

This gives a powerful selection mechanism, with the same expressiveness as the formula of monadic second order logic of the tree with one free variable, or monadic Datalog [Gottlob and Koch, 2004].

We consider also a restriction of the SA where a position p in a term t is selected if the sequence of symbols on the path from p to the root of t belongs to a given regular (word) language. More precisely, a selection automaton $\mathcal{A} = \langle \Sigma, Q, F, S, \Delta \rangle$ is called *prefix* if Q contains two special states: q_0 (universal state) and q_s (selection state), $F \subseteq Q \setminus \{q_0\}$, $S = \{q_s\}$, and Δ contains $f(q_0, \ldots, q_0) \to q_0$ and $f(q_0, \ldots, q_0) \to q_s$ for all $f \in \Sigma$, and some other transition rules of the form $f(q_1, \ldots, q_n) \to q$ where $q \in Q \setminus \{q_0, q_s\}$ and there exists exactly one $i \leq n$ such that $q_i \neq q_0$. Intuitively, assume that we are given a finite automaton $\mathcal{B}$ defining the strict prefixes of selected positions. Then q_s is the initial state of $\mathcal{B}$, F is the set of final states of $\mathcal{B}$, and for all $a \in \Sigma_n$, Δ contains n rules $a(q_0, \ldots, q_0, q', q_0, \ldots, q_0) \to q$ for each transition $q' \xrightarrow{a} q$ of $\mathcal{B}$. Note that with this definition, the root position is always selected by a prefix selection automaton.

Controlled Term Rewriting Systems

We propose a formalism that extends standard TRS presented in Section 1 by restricting the possible rewrite positions to positions selected by a given SA. Formally, a *controlled term rewriting system* (CTRS) $\mathcal{R}$ over Σ is a finite set of *controlled rewrite*

rules of the form $\mathcal{A} : \ell \to r$, made of a SA $\mathcal{A}$ over Σ and a *rewrite rule* $\ell \to r$ such that $\ell \in \mathcal{T}(\Sigma, \mathcal{X}) \setminus \mathcal{X}$ and $r \in \mathcal{T}(\Sigma, vars(\ell))$.

The rewrite relation defined by a CTRS $\mathcal{R}$ is the usual rewrite relation of TRS, restricted to the positions selected by the SA of the rewrite rules. More precisely, a ground term s rewrites to t in one step by a CTRS $\mathcal{R}$, denoted by $s \xrightarrow[\mathcal{R}]{} t$, if there exists a controlled rewrite rule $\mathcal{A} : \ell \to r \in \mathcal{R}$, a position $p \in sel(\mathcal{A}, s)$, and a substitution σ such that $s|_p = \sigma(\ell)$ and $t = s[\sigma(r)]_p$. The CTRS such that all selection automata associated to rules are prefix are called *prefix controlled rewrite systems* (pCTRS).

The *innermost strategy*, presented in Section 3.1, can be expressed in controlled rewriting for left-linear TRS, because the set of normal forms for such TRS are recognizable by tree automata.

The set of positions selected for the *context-sensitive strategy* can be defined by SA, *i.e.* context-sensitive rewriting is a particular case of controlled rewriting. It is a strict subcase because with the context-sensitive strategy, the root position is always rewritable whereas this is not the case for controlled rewriting.

Flat and Ground Controlled Rewrite Rules

It turns quickly out that CTRS are too powerful: they can simulate computations of Turing machines even under strong syntactical restrictions (recall that a signature is unary when all its symbols have arity 0 or 1).

Theorem 55 [Jacquemard et al., 2011b]

Reachability is undecidable for flat CTRS over unary signatures and for ground CTRS.

These results contrast with the case of plain rewriting because both classes of linear and flat TRS and ground TRS preserve regular tree languages, and in both cases reachability is decidable in PTIME. Reachability is decidable when restricting to words (unary signature), prefix control and flat controlled rewrite rules.

Theorem 56 [Jacquemard et al., 2011b]

Reachability is decidable in PSPACE for flat non-collapsing pCTRS over unary signatures.

However, flat pCTRS over unary signatures do not preserve regularity (see Section 3.3).

Monotonic Controlled Rewrite Rules

We consider in [Jacquemard et al., 2011b] some larger class of tree languages strictly larger than regular languages: the *context-free* tree languages defined in Section I.2.5 and *context-sensitive* tree languages defined as follows. A TRS is called *monotonic* if it is linear and all its rules $\ell \to r$ are such that $vars(r) = vars(\ell)$ and $|\mathcal{P}os(\ell)| \leq |\mathcal{P}os(r)|$.

Let us call a tree grammar $\mathcal{G} = \langle \mathcal{N}, S, \Sigma, P \rangle$ *context-sensitive* (CSTG) when all the production rules of P are monotonic. Membership is PSPACE-complete and emptiness undecidable for CSTG (see [Jacquemard et al., 2011b]).

Theorem 57 [Jacquemard et al., 2011b]

Given a CSTG $\mathcal{G}$ and a monotonic CTRS $\mathcal{R}$, one can construct in PTIME a CSTG generating the closure $post^*_{\mathcal{R}}(\mathcal{L}(\mathcal{G}))$, and whose size is linear in the size of $\mathcal{G}$ and $\mathcal{R}$.

It follows that reachability is decidable (in PSPACE) for monotonic CTRS. Moreover, with flat and monotonic CTRS over unary signatures (*i.e.* CTRS over words with rules of the form $\mathcal{A} : a(x) \to b(x)$) it is possible to simulate *linear bounded automata* [Kuroda, 1964] (Turing machines computing on a tape of fixed length). Therefore, reachability is NLINSPACE-complete and model checking is undecidable for these CTRS.

CF Controlled Rewrite Rules

The controlled rewrite systems have been introduced in the case of word rewriting, see [Sénizergues, 1993] for a survey. The related *conditional context-free grammars* [Dassow et al., 1997] can be redefined in our settings as word grammars (tree grammars over unary signatures), whose production rules are CF controlled rewrite rules (CF rewrite rules are defined in Section 1.3) and derivations are defined using the controlled rewrite relation. It is shown in [Dassow et al., 1997] that the class of languages of conditional CF grammars without collapsing rules coincide with context-sensitive (CS) word languages. Hence, it also holds that reachability is PSPACE-complete and model checking is undecidable for CF non-collapsing CTRS over unary signatures.

Some other former results in the case of words imply that the above lower bounds still hold when control is limited to prefix SA: it is shown in [Penttonen, 1974] that every CS word language can be generated by a CS grammar with production rules of the form $A(B(x)) \to A(C(x))$, $A(x) \to B(C(x))$, $A(x) \to a(x)$ (where A, B, C are non-terminal and a is a terminal). It follows that every CS word language is the closure of a constant symbol under a CF non-collapsing pCTRS (over a unary signature).

Theorem 58 [Jacquemard et al., 2011b]

Reachability is PSPACE-complete and regular model checking undecidable for CF non-collapsing pCTRS over unary signatures.

Recursive Control

We consider in [Jacquemard et al., 2011b] a relaxed form of the prefix control on rewrite rules, where the selection is done by considering the term in input modulo the rewrite relation. A *recursive* pCTRS $\mathcal{R}$ is defined as a pCTRS. In order to define formally the rewrite relation, let us recall first that in the runs of a prefix SA $\mathcal{A}$, the states below a selection state q_s are all universal (q_0), hence we can have any subterm at a selected position (only the part of the term *above* the selected position matters). Following this observation, we say that the variable position p in a context $C[x_1]$ over Σ is *selected* by the prefix SA $\mathcal{A}$ if p is selected in $C[c]$ where c is an arbitrary constant symbol of Σ_0. A term s rewrites to t in one step by a recursive pCTRS $\mathcal{R}$, denoted by $s \xrightarrow[\mathcal{R}]{} t$, if there exists a controlled rewrite rule $\mathcal{A} : \ell \to r \in \mathcal{R}$, where $\mathcal{A}$ is a prefix SA, a substitution σ, a position $p \in \mathcal{P}os(s)$, such that $s|_p = \sigma(\ell)$, $t = s[\sigma(r)]_p$, and moreover there exists a context $C[x_1]$ such that $C[x_1] \xrightarrow[\mathcal{R}]{*} s[x_1]_p$ and the position of x_1 is selected in $C[x_1]$ by $\mathcal{A}$. This definition is well-founded because of the restriction to prefix control (remember that the root position is always selected by prefix SA).

Example 23 *Let $\Sigma = \{a, b, c, d, \bot\}$ be a unary signature, and let $\mathcal{R}$ be the flat recursive pCTRS containing the rules $\mathcal{C}_1 : a(a(x)) \to b(x)$, and $\mathcal{C}_2 : c(x) \to d(x)$, where the SA $\mathcal{C}_1$ selects the position after a prefix aa, and $\mathcal{C}_2$ selects the position after a prefix $aaaa$. Then we have with $\mathcal{R}$ (we omit the parentheses and the tail $\bot$, and underline the part of the term which is rewritten)*

$$aa\underline{aa}c \xrightarrow[\mathcal{R}]{} aab\underline{c} \xrightarrow[\mathcal{R}]{} aabd$$

Note that for the last step, we have use the fact that $aaaa \xrightarrow[\mathcal{R}]{} aab$, i.e. that there exists $C[x_1] = aaaa(x_1)$ with $C[x_1] \xrightarrow[\mathcal{R}]{} aab(x_1)$ and the position of x_1 in $C[x_1]$ is selected by $\mathcal{C}_2$. The last rewrite step would not be possible if $\mathcal{R}$ would not be recursive, because aab is not a prefix admitted by $\mathcal{C}_2$. ◇

With this recursive form of prefix controlled rewriting, we obtain a regularity preservation result.

Theorem 59 [Jacquemard et al., 2011b]

Regular model-checking is decidable in EXPTIME for linear and right-shallow recursive pCTRS.

The proof is based on the construction of an alternating tree automata with ε-transitions (such automata recognize regular tree languages).

Tree Transducers with Regular Look-Ahead

Top-down tree transducers with regular look-ahead [Engelfriet, 1976] (which have been used to represent XML transformations, see *e.g.* [Engelfriet et al., 2009]) are an extension of top-down tree transducers, where a transition can be fired provided that the current subtree belongs to some given regular tree language. This is similar to our notion of control for rewrite systems. A notable difference however is that the transducers transform terms in one (top-down) pass, whereas we consider here the terms computed by an arbitrary iteration of controlled rewrite rules.

Perspective: Verification of XML Updates and ACPs

One of our motivations for studying CTRS is the specification, by controlled rewrite rules, of XML transformations whose positions of application (*target* positions, using the vocabulary of [Robie et al., 2011]) are defined by some formalism enabling unary queries, like Monadic Datalog or Core XPath [Gottlob and Koch, 2004]. In particular, XPath serves as a sublanguage in the W3C recommendation for XQuery Update [Robie et al., 2011]. The navigational core of XPath corresponds, roughly, to first order logic with two variables (*wrt* unary queries)) and it is hence less expressive than SA (which corresponds to monadic second order logic with one free variable). The prefix SA correspond to a fragment of Regular XPath [Marx, 2004] which may be interesting in practice.

The results presented above may be seen as preliminary work towards an extension of the approach presented in Sections 2.4 and 2.5 for the static analysis of the iterations of XML updates rules with target selection mechanisms. These results are in particular limited to ranked term rewriting whereas the rewrite rules used in Sections 2.4, 2.5 are hedge rewrite rules.

Some previous work already consider transformation formalisms close to hedge rewriting extended with XPath selection. For instance, in [Jacquemard and Rusinowitch, 2010] we show that reachability is undecidable for sets of rewrite rules of type REN and RPL_1 (defined in Figure 11) extended with selection by a fragment of XPath. Some similar undecidability results can also be found in [Fundulaki and Maneth, 2007] and [Moore, 2011]. A decidable case is also presented in [Moore, 2011]: roughly, reachability for a controlled extension of ACP rules close to the definition of Section 2.5, without delete rules (hence monotonic controlled rewrite rules, in the above sense).

Note also that, in the definition of the set of selected positions $sel(\mathcal{A}, t)$, for a SA $\mathcal{A}$, it is required that t is recognized by $\mathcal{A}$ in order to select positions. Hence, in the context of ranked terms, the CTRS are more general than the PGHRS defined in Section 2.5 for the DTD based specification of update ACPs, hence the undecidability result of Theorem 51 also holds for controlled rewrite rules of type rename or replace.

Perspective: Regularly Controlled Rewriting

In Section 2.4, we considered the application, to XML documents, of arbitrary sequences of update primitives (amongst a given finite set of primitives represented by a PHRS). Our results can be applied to the verification of set of allowed updates primitives, see in particular Section 2.5 and application to the verification of access control policies for updates.

In the XQuery Update Facility [Robie et al., 2011] however, an update is not any sequence of update primitives. It is an XQuery expression u, containing update primitives, which is applied in several phases to a given document t. In a first phase, u is converted into a sequence w of primitive updates to be applied to t (the *pending list* generated by u over t). Then in a second phase, the sequence w is analyzed (sanity check) and the order of its elements is possibly modified (according to a priority ordering for the update primitives), giving a new sequence w'. Afterward, in a final phase, w' is applied to t, returning the updated document t'.

In [Benedikt and Cheney, 2009], an finite (approximative) representation is proposed for the set of the pending lists generated by an update expression u for all the trees in an input type L_{in} (an HA language). It consists in a regular expression Ω over the alphabet of all possible update primitives, called the *effect expression* in [Benedikt and Cheney, 2009]. The alphabet of update primitives corresponds roughly to the set of rewrite rules presented in Figure 11 in Section 2.3. Note that for a given Σ, a fixed HA $\mathcal{A}$, and a fixed number n for RPL, this set is finite. Let us call it $\mathcal{R}/\mathcal{A}$. Following this approach, the effect of the application of u to the trees in L_{in} can be represented in our settings, by the refined closure

$$post^{\Omega}_{\mathcal{R}/\mathcal{A}}(L_{\mathsf{in}}) = \{h' \in \mathcal{H}(\Sigma) \mid \exists h \in L, \exists r_1 \dots r_n \in \Omega, h \xrightarrow[r_1]{} \dots \xrightarrow[r_n]{} h'\}.$$

The computation of $post^{\Omega}_{\mathcal{R}/\mathcal{A}}(L_{\mathsf{in}})$ (given a finite PTRS $\mathcal{R}/\mathcal{A}$, Ω a regular expression over $\mathcal{R}/\mathcal{A}$ and an HA $\mathcal{A}_{\mathsf{in}}$ recognizing L_{in}) is therefore an interesting problem, apparently related to the study of languages of context-free grammars with regulated rewriting [Dassow et al., 1997]. It has been studied by Ryma Abassi during an internship in 2010. It is related to the study of languages of context-free grammars with regulated rewriting [Dassow et al., 1997]: languages of words generated by a given CF grammar $\mathcal{G}$ following a derivation which is a sequence of production rules of $\mathcal{G}$ in given regular language R. These languages range between context-free and context-sensitive (when collapsing production rules are excluded), and membership is decidable but emptiness undecidable.

3.3 CF Unranked Tree Languages

It is known that the transformations of unranked ordered trees go quickly out of HA (*i.e.* regular) languages, because they can transform a HA language into a non-HA language [Vianu, 2001]. We have already observed some examples of this fact in Sections 2.2 and 2.3. There, we saw that for various cases of HRS which do not preserve HA language but transform a HA language into a $\mathsf{CF\text{-}HA}$ language. For

instance, the language $\{g(a^n\, c\, b^n) \mid n \geq 0\}$ of Example 17 is a CF-HA language obtained by rewrite closure with the rule $g(x) \to g(axb)$. In can also be interesting to consider unranked tree languages more general than CF-HA, with context-free constructions in the horizontal or the vertical dimensions, or combining both. For instance, in page 87 (after Theorem 44), we have seen how to produce a language of trees of the form $\underbrace{g(g(\ldots g(}_{n} f(a^n c))))$, by closure with the hedge rewrite rule $f(x) \to g(f(ax))$.
This kind of generalized CF unranked ordered tree languages can also be useful for the study of XML updates, as presented in the next examples.

Controlled PHRS

As explained in Section 3.2, for XML standards like XQuery update [Robie et al., 2011], a path specification language XPath is used in order to define the position where the updates can be applied. Let us add some context control, with selection automata (generalized to HA) to the PHRS rules presented in Section 2.3, Figure 11, in order to model this feature of XQuery updates. The controlled version of PHRS is called CPHRS.

We have seen in Theorem 55 that, even in the case of words (every position in the tree has zero or one child) flat controlled rules of the form REN and $\mathsf{DEL_s}$ are sufficient to simulate Turing machines. Restricting to prefix control, we define a class called pCPHRS which does not preserve HA languages. Consider for instance the pCPHRS containing the 4 following rules of type REN (in the rewrite rules, we use an informal description of the languages of the prefix selected, instead of giving explicitly the prefix SA):

$$\begin{array}{llllll} \text{no } a', \text{ no } a & : & c(x) & \to & a'(x), \\ \text{no } a', \text{ no } a & : & a'(x) & \to & a(x), \end{array} \qquad \begin{array}{llllll} \text{exactly one } a' & : & d(x) & \to & b'(x), \\ \text{no } a', \text{ no } b', \text{ no } b & : & b'(x) & \to & b(x). \end{array}$$

The intersection of the HA language of trees of the form $a(\ldots a(b(\ldots b(b)))$ with the closure of the HA language of trees of the form $c(\ldots c(d(\ldots d(d)$ by the above four prefix controlled rewrite rules, is $\{\underbrace{a(a(\ldots a}_{n}\underbrace{(b(\ldots b(b)}_{m} \mid n \geq m\}$. This set is not a HA and neither a CF-HA language. It is easy to find variants of the above examples, using rules of the form $\mathsf{INS_{first}}$ or $\mathsf{INS_{before}}$ instead of REN, and producing *e.g.* set of tree of the form $\underbrace{a(a(\ldots a}_{n}(b^m)$ with $n \geq m$.

General CF Tree Grammars

During an internship at LSV [Boiret, 2010] (with Luc Segoufin and myself), Adrien Boiret has studied several possible definitions of context-free unranked ordered tree languages. Our goal was to find a formalism which permits to express

1. all CF languages of trees of bounded rank (as defined in Section I.2.5),

2. the image of every CF binary tree language under the inverse of an encoding from unranked into ranked trees, like the one defined in Section 2.1.

The difficulty, compared to the definition of context-free ranked tree grammars (see Section I.2.5), is to decompose the list of children of one position, without knowing in general the number of children. The proposal of [Boiret, 2010] for this problem is to provide, access to the first child of a position, with an operator *head*, and to the rest of the children, with the operator *tail*.

A *context-free unranked tree grammar* is a tuple $\mathcal{G} = \langle \mathcal{N}, S, \Sigma, P \rangle$ where $\mathcal{N}$ is a finite set of *non-terminal* symbols (without arities), $S \in \mathcal{N}$ is the *axiom* of $\mathcal{G}$, Σ is a finite alphabet disjoint from $\mathcal{N}$ (its elements are also called *terminal* symbols), and P is a set of *production rules*, of the form $N(x) \to r$ where $N \in \mathcal{N}$, x is a variable and $r \in \mathcal{O}\big(\Sigma \cup \mathcal{N} \cup \{head, tail\}, \{x\}\big)$. We assume moreover that all the subtrees of r headed by *head* and *tail* have the form $head(x)$ and $tail(x)$.

The derivation relation $t \xrightarrow[P]{*} s$ induced by a unranked tree grammar is the reflexive and transitive closure of the (hedge) rewrite relation defined by P, with, moreover, after every rewrite step, a normalization under the simplification axioms: $head(a(x)\, y) \to a(x)$ and $tail(a(x)\, y) \to y$ for all $a \in \Sigma \cup \mathcal{N}$. We also assume an alternative strategy, call *non-discriminative* in [Boiret, 2010] where two other normalization rules are applied to the empty hedge: $head() \to ()$ and $tail() \to ()$. The *language* of the tree grammar $\mathcal{G}$ is defined as $\mathcal{L}(\mathcal{G}) = \{t \in \mathcal{O}(\Sigma) \mid S \xrightarrow[P]{*} t\}$. Note that the symbols *head* and *tail* are excluded from the language. It means that they must have been eliminated the simplification axioms. Under the discriminative strategy, an occurrence of *head* or *tail* applied to the empty hedge is blocking.

It is shown in [Boiret, 2010] that the discriminative formalism is more powerful than the non-discriminative one, and that the non-discriminative unranked tree grammars fulfill the two above goals (1) and (2) (*i.e.* they generalize the ranked tree context-free grammars). Unfortunately, emptiness appears to be undecidable for this formalism.

Theorem 60 [Boiret, 2010]

The emptiness problem is undecidable for non-discriminative context-free unranked tree grammars.

This result is shown by a reduction of the reachability problem for 2 counters machines. Moreover, it is also shown that restricting to a top-down (outermost) strategy for the application of the production rules is a real limitation in expressiveness, still satisfying the goals (1) and (2), but also still with an undecidable emptiness problem.

The question of the definition of a decidable and sufficiently expressive class of CF unranked ordered tree languages (more general than CF-HA languages) is still not totally clear, since many options are possible, combining the horizontal and vertical dimensions in trees. An alternative definition would be to consider only the images of CF binary tree languages (following the definition of Section I.2.5) under the inverse of

a transformation of unranked ordered trees into binary trees (*e.g.* the transformation presented in Section 2.1). It would be interesting to relate the languages defined this way with the CF-HA languages. Moreover, as suggested by Sebastian Maneth, an interesting approach for defining decidable classes of languages above CF-HA could be also to consider the inverse images of the hierarchies of ranked tree languages defined in [Engelfriet, 1981] by the composition of top-down tree transducers.

Perspective: Atomic CF Tree Grammars

A subclass of the above context-free unranked tree grammars with a decidable emptiness problem is defined by the case where the production rules do not contain any occurrence of *head* and *tail*. Let us call *atomic unranked tree grammars* this subclass. This restriction means that, in the grammar derivation, the lists of siblings cannot be decomposed. Therefore, atomic unranked tree grammars do not satisfy the goal (1). Nevertheless, we believe that they can be useful in for verification of XML updates and XML read/write access control policies. An interesting question in this context is whether the update PHRS rules of Figure 11 (Section 2.3), extended with prefix control, preserve the languages of atomic unranked tree grammars.

Alternatively, we could also consider a control of the subtrees below the selected rewrite positions, similar to the tree transducers with look-ahead[Engelfriet, 1976]). More precisely, in this case, a rewrite position p is selected in a tree t if the subtree of $t|_p$ belong to a fixed HA language. This restriction can be defined by a condition on selection automata symmetric to the prefix automata. The idea behind the above condition, for modeling updates, for instance in order to express that a patient will be updated only if its records contain some special information. Whether the atomic unranked tree grammars capture the closure of HA language under rewriting with such a strategy is also an interesting question.

III

Perspectives

We have presented many perspectives in the previous pages, some of them are direct extensions of former works, some could be the subject of longer studies. Let us conclude this document with a discussion on two particular topics, which could be consider as mid-term research goals (for instance, the subjects of master or PhD thesis). The first section below summarizes several ideas regarding models tree automata with constraints, presented in Parts I and II. The second section is more prospective and aims at establishing some bridges between too independent family of models: tree automata (and logics) with equality constraints and the formalisms for data trees.

1 Generalized Constraints for Tree Automata

The models of extended tree automata presented in Part I are limited to testing equalities, *i.e.* isomorphism between subtrees in the tree in input. It could be interesting for several reasons to study extensions of tree automata with other kind of constraints (either local or global). We present below some examples of such extensions that could be worth studying.

1.1 Equality Modulo Equational Theories

It is sometimes useful to interpret equalities between trees not simply as tree isomorphism but rather as equality modulo equational theories (*i.e.*, in the case of terms, as the binary relation $\overset{\mathcal{R}}{\longleftrightarrow}$ for some TRS $\mathcal{R}$).

For instance, in Section I.2.5 (Theorem 20), we saw that relaxing the equality relation of $\mathsf{VPTA}^{=}$ into structural equality gives a model $\mathsf{VPTA}^{\equiv}$ enjoying better closure properties. Structural equality can be interpreted as equality modulo a TRS that "wipes out" the function symbols into blank symbols $\bot_n$ with rules such as $f(x_1, \ldots, x_n) \to \bot_n(x_1, \ldots, x_n)$. This relaxation permits us to gain closure under intersection and complementation. More generally, we consider in a long version of [Barguñó et al., 2010] (see Section I.3.4), currently in submission, equality and disequality constraints interpreted modulo a flat equational theory (this includes in particular structural equality) both for global and the local constraints between brother subtrees.

It is has been observed that in general, isomorphism constraints do no combine well with closure under rewriting. In other terms, the closure of the languages of tree automata with this kind of constraints can generally not be captured by tree automata with constraints, see Section II.1.4. An exception is presented in Section II.3.1 (Theorem 53), thanks to a use of the innermost strategy. We obtain in general better results for computing the closure by a rewrite system when considering equality modulo the same rewrite system, see the discussions in Sections I.2.4 and II.1.4. For instance, the TADE of Section I.2.4 are defined as the union of a set of automata clauses (with equational constraints, of type TAD) and a set of equational clauses. It means that the equational constraints of the TAD part are interpreted modulo the theory defined by the equational part. The equational theories for which decidability of emptiness can be achieved for TADE is however limited.

A more specialized approach focuses on the closure of tree languages or modulo associativity and commutativity (AC). We have already mentioned as a perspective in Section I.2.7 the extension of the TADE model modulo AC, using AC-paramodulation techniques [Nieuwenhuis and Rubio, 2001] for the decision procedures.

Moreover, we have seen in Section I.2.1 that CF-HA are equivalent (via the flattening operations defined in Figure 10) to the class of closure of regular ranked term languages modulo associativity, and the Presburger automata (PTA) presented in Section I.2.6 correspond to the class of closure of regular ranked term languages modulo associativity and commutativity. A challenging perspective could be to study the extension of CF-HA and PTA with global constraints of equality and disequality modulo A and AC.

1.2 Ordering and other Constraints

We have outlined in Section I.1.5, the interest of the construction of normal form tree automata in the context of automatic theorem proving. The procedure of [Bouhoula and Jacquemard, 2008] handles constrained rewrite rules between constructors of the form $c \Rightarrow \ell \to r$, where the constraint c is a conjunction of atoms $P(t_1, \ldots, t_n)$, P being a predicate with a fixed interpretation on ground constructor terms, and $t_1, \ldots, t_n$ being terms built with constructor symbols and variables of ℓ. Therefore, the automata recognizing the normal forms for such constrained TRS may contain arbitrary constraints (roughly, the negation of constraints found in the rewrite rules).

We identify in [Bouhoula and Jacquemard, 2008] some cases of constrained TRS such that the associated normal form tree automata belong to decidable subclasses of TAC. It would interesting to establish emptiness decidability results for some other tree automata models, with different kinds of local constraints, defined by predicates interpreted on ground terms.

One example of other constraints is the equality of height of terms. A tree automata model with similar constraints is studied in [Habermehl et al., 2010] where it is applied to the analysis non-recursive C programs manipulating balanced tree-like data structures. We also use such tree automata in [Bouhoula and Jacquemard,

2008] for proofs on a specification of powerlists – lists of length 2^n, for $n \geq 0$, whose elements are stored in the leaves of a balanced binary tree.

One could also consider constraints based on *reduction orderings* [Dershowitz and Jouannaud, 1990]. As outlined in Section I.1.5, adding such ordering constraints to rewrite rules permits ones inductive theorem proving on complex data structures like *ordered lists*, and permits ones also to transform a non terminating TRS into an equivalent orientable theory (containing rules with ordering constraints), by application of constrained completion techniques [Kirchner et al., 1990]. To my knowledge, tree automata with ordering have not been studied yet, and designing emptiness decision procedure for such tree automata seems a challenging problem.

1.3 Separated Constraints

A recent proposal by Christoph Weidenbach and Jochen Eisinger, which is the subject of a current ongoing joint work, is to restrict the definition of constraints for tree automata to terms over a distinguished signature Σ. The proposal is to consider tree automata with very general local constraints, defined as first-order formula of the form $\phi(\pi_1, \ldots, \pi_n)$ where ϕ is built over a set of function symbols Σ, and a set of predicates $\mathcal{P}$, and the free variables π_1,...,π_n are positions (exactly like the positions in equality and disequality constraints $\pi = \pi'$, $\pi \neq \pi'$ of the TAC of Section I.1). Given a fixed first order interpretation structure $\mathfrak{A}$ for Σ and $\mathcal{P}$, with domain $dom(\mathfrak{A})$, constraints as above can be evaluated at a position p of a ground term t, by replacing in ϕ every π_i $(1 \leq i \leq n)$ by the subterm $t|_{p \cdot \pi_i}$, providing that this subterm belongs to $\mathcal{T}(\Sigma)$. This permits the constrained tree automata to compute on ground terms of $\mathcal{T}(\Sigma \cup \Sigma')$, using this interpretation of the constraints at every computation step.

The difficulty is of course the decidability of emptiness for these tree automata, and some assumptions on the first-order structure $\mathfrak{A}$ are necessary. A possibility could be to assume the decidability of the first-order theory of $\mathfrak{A}$, and the existence of a bound B on the index of the equivalence $\equiv$ defined on tuples of elements of $dom(\mathfrak{A})$, roughly, by the satisfaction of the same formula. The goal is to enable a pumping argument implying the decidability of emptiness: in a term of $t \in \mathcal{L}(\mathcal{A})$ sufficiently large (according to a bound which depends on B) a replacement of equivalent ($wrt \equiv$, after interpretation of the terms in $\mathfrak{A}$) subterms is possible, reducing t into a smaller term of $\mathcal{L}(\mathcal{A})$. Note that assumptions as above should probably not hold for the equality and disequality constraints of the TAC of section Section I.1. However a good candidate for $dom(\mathfrak{A})$ could be the automatic structures of [Blumensath and Grädel, 2000], including Presburger arithmetic.

An interesting constrained tree automata model would be an extension of $\mathsf{TA}_{\neq}$ with constraints as above: it would permit us to characterize the languages of ground normal forms of rewrite systems with constrained rules of the form $\phi(x_1, \ldots, x_n) \Rightarrow \ell \to r$, where $\ell, r \in \mathcal{T}(\Sigma', \mathcal{X})$, ϕ is a first order formula over Σ and $\mathcal{P}$ and $x_1, \ldots, x_n \in vars(\ell)$. We believe that, under assumptions similar to the above ones, a adaptation of the emptiness decision procedure presented in Section I.1.3 could be applied to this extension.

2 Data Trees and Tree Isomorphisms

In Section I.3.6, we discuss the application of results of Part I to reasoning tasks for XML documents which are represented by trees labeled with a finite alphabet. Many theoretical works on XML consider a more realistic model of XML document as *data tree*, whose positions carry both a label from a finite alphabet and a data value from an infinite domain. Some models of automata and logic working on such trees have been proposed, [Bojańczyk et al., 2009; Bojanczyk et al., 2006; Demri and Lazić, 2009; Jurdzinski and Lazic, 2007; Bouyer et al., 2001; Neven et al., 2001], with the ability to compare the data values. One example is $\mathsf{FO}^2[+1, \sim]$, the first order formulas with 2 variables and the navigation predicates $S_\downarrow$ and $S_\rightarrow$ defined Section I.3.6 and a predicate $\sim$ for testing the equality of data values. This logic is decidable on data trees [Bojańczyk et al., 2009] (with a 3NEXPTIME complexity), as well as its extension $\mathsf{EMSO}^2[+1, \sim]$ with existential monadic second-order quantifiers.

Comparing the models for data tree with the formalisms presented in Part I, *e.g.* $\mathsf{TAGC}[\approx, \not\approx, |.|_{\mathbb{N}}, \|.\|_{\mathbb{N}}]$ and $\mathsf{EMSO}[+1, \approx, \not\approx, \mathbb{N}]$, testing isomorphisms on subtrees, is a difficult question, which has probably no simple or systematic answer. We propose below some reflections on this problem.

2.1 Equality of Data Values and Subtrees

Encoding Data Values as Subtrees

The simplest approach for relating the formalisms for data trees and the formalisms testing isomorphism of subtrees is to encode the data values as trees. For instance, one can represent the natural numbers by terms built with the symbol 0 and successor s, and transform every data tree of $\mathcal{O}(\Sigma \times \mathbb{N})$ into an unranked tree of $\mathcal{O}\big(\Sigma \uplus \{s, 0\}\big)$ by adding a new child to every position, for carrying the tree representation of its data value. More precisely, this is done by the mapping τ defined recursively by $\tau\big(\langle a, n\rangle(t_1, \ldots, t_n)\big) = a(\tau(t_1), \ldots, \tau(t_n), s^n(0))$, where $a \in \Sigma$ and $n \in \mathbb{N}$. Note that the codomain of τ is regular. Applying $\approx$ constraints of TAGC or equality constraints of TAC to the auxiliary child encoding data values permits us to simulate the predicate $\sim$ on data values.

However, some properties expressible in FO^2 cannot be expressed with TAGC via τ. For instance, this is the case of the *inclusion constraints* presented in page 64, expressing that for all position x labeled with a there exists a position y labeled with b with the same data value, in FO^2 $\forall x \exists y\ a(x) \Rightarrow (b(y) \wedge x \sim y)$. Conversely, the equality between subtrees, *i.e.* the TAGC constraint (and predicate of EMSO) $\approx$ is not expressible in $\mathsf{MSO}[+1]$ on trees [Comon et al., 2007], hence it is neither expressible in $\mathsf{EMSO}^2[\sim, +1]$ (data values cannot help for this purpose).

Another interesting formalism for a comparison is the alternating register automata over data trees proposed in [Jurdzinski and Lazic, 2007]. These automata (called ATRA^1) use one register for data values: at every position p of computation, the data value at p can be stored in the register or compared to the data value currently in the register. Emptiness is decidable for these automata (though

non primitive recursive) and undecidable when there is more than one register. The the ATRA^1 have been introduced as a decision tool for an alternation-free modal μ-calculus over data trees generalizing to trees the extension of the linear temporal logic LTL with freeze quantification (these quantification play the role of the above registers) [Demri and Lazić, 2009]. We have seen in Section I.3.2 that the RTA (which are equivalent in expressiveness to the positive $\mathsf{TAGC}[\approx]$) can be presented as sets of Horn clauses with rigid variables. Intuitively, these rigid variables can be seen as registers storing subtrees, which can be used only once for writing, but an unbounded number of times for reading. Hence, roughly, the ATRA^1 have one register which can be rewritten and the RTA have several non-rewritable registers. Studying the expressiveness of tree automata with global constraints from this angle could be an interesting subject, for instance for identifying a corresponding fragment of a modal logic over data trees (with freeze quantifier).

Besides the problem of expressiveness, it would also be interesting to study the complexity of emptiness decision for TAGC computing on the encoding of data trees. Indeed (as already noted in Section I.3.6), in this case, all the positions of subtrees tested for isomorphism by an automaton during a computation are incomparable (*wrt* the prefix ordering). We believe that such a strong restriction could permit us to decrease dramatically the complexity upper bounds for emptiness decision.

Encoding Subtrees as Data Values

We may also wonder whether the decidable formalisms for data trees could be used in order to improve the expressiveness of tree automata of Part I. As noted in Section I.3.6, reasoning on integrity constraints in presence of HA is more complex when equality is interpreted as subtree isomorphism than when it is interpreted as data equality. The reason is that data values in the different positions of a data tree are independent, whereas there are dependencies between subtrees. Hence, some restrictions are necessary in order to reduce the problems of automata with equality constraints to problems for formalisms for data trees.

For instance, we could require that the data at a position p in a tree $t \in \mathcal{O}(\Sigma \times \mathbb{N})$ is uniquely associated to the subtree $\mathit{erase}(t|_p)$ obtained by erasing the data values. More formally, t is required to be such that every two positions p_1 and p_2 in $\mathcal{P}os(t)$ have the same data value iff $\mathit{erase}(t|_{p_1}) = \mathit{erase}(t|_{p_2})$ (*). Note that it is a strict restriction on data trees. Assuming this restriction, we can express in EMSO^2 the existence of a run of a given TAGC. However, the property (*) is not expressible in MSO^2 and we do not know whether $\mathsf{FO}^2[\sim, +1]$ is decidable or not on the subset of data trees following this restriction.

We can observe that when considering the model of binary trees, in a variant of FO^2 where the two relations $S_\downarrow$, $S_\rightarrow$ are replaced by successor functions, $\mathsf{FO2}[\sim, +1]$ is undecidable on the data tree satisfying the above restriction (*). We can indeed encode any finite grid $\mathbb{N} \times \mathbb{N}$ as a binary tree such that for all position x, $x.1.2 \sim x.2.1$.

2.2 Tree Automata with Global Constraints for Data Trees

Another approach could be to use the TAGC to compute on data trees, the constraints being interpreted as value equality, instead of subtree equality. More precisely, given a TAGC $\mathcal{A}$, a data tree $t \in \mathcal{O}(\Sigma \times \mathbb{N})$, and a run r of the underlying TA on $erase(t)$, $r \models q \approx q'$ (respectively $r \models q \not\approx q'$) if for all different positions $p, p' \in \mathcal{P}os(t)$ such that $r(p) = q$ and $r(p') = q'$, $data(t(p)) = data(t(p'))$ holds (respectively $data(t(p)) \neq data(t(p'))$), where $data(\langle a, n\rangle) = n$.

A recent paper [David et al., 2011], introduces tree automata extended with a conjunction of (global) constraints of two kinds. Constraints of the first kind are called *linear data constraints*, and are linear integer constraints over variables similar to $|a|$ and $\|a\|$ defined in Section I.3.5, except that they refer to the data and not the subtrees, and that a is a symbol of Σ and not a state of the automaton. Hence the interpretation of these linear constraints in [David et al., 2011] depends only on a data tree t and not on a run of the automaton: $|a|$ is the number of positions labeled by a in t and $\|a\|$ is the cardinality of the set of data values found at positions labeled by a in t. The second kind of constraints are set constraints over the sets of data values on the position labeled by a particular letter of Σ. The emptiness is shown decidable in NP for these automata [David et al., 2011] using integer linear programming techniques.

The same techniques should be applicable to TAGC$[|.|_{\mathbb{Z}}, \|.\|_{\mathbb{Z}}]$. Note that ([David et al., 2011]) a key constraint, for state q, can be expressed by $|q| = \|q\|$. Whether is could be applicable also to the whole class TAGC$[\approx, \not\approx, |.|_{\mathbb{Z}}, \|.\|_{\mathbb{Z}}]$ (on data trees) is an interesting question.

2.3 Data Tree Rewriting

One other perspective related to the Section II.2 on unranked tree rewriting is to add constraints to rewrite rules in order to handle data values. In [Bouajjani et al., 2007], a notion of constrained data word rewriting system is proposed. The constraints are expressed in a first-order logic over data words. This framework is applied in [Bouajjani et al., 2007] to the verification of programs with unbounded control structures and unbounded data domains

To my knowledge, there is no generalization of term rewriting to data tree rewriting yet. This could be useful for representing XML transformations that take into account data values. In could be interesting to define a formalism of data tree rewriting systems and study the closure, by such systems, of unranked tree automata with global constraints on data values in the context of typechecking data tree transformations.

2.4 Generalized Global Constraints for Tree Automata

The constraints of TAGC are universally quantified, and if they are well-suited to express key or denial constraints, they cannot express constraints defined with an alternation of quantifiers, such as inclusion constraints: for all position p of type q,

there exists a position p' of type q' such that the subtrees at p and p' are the same. The emptiness decision proof technique of Section I.3.4 does not work for this kind of constraints.

TA with Global Monadic First Order Constraints

One possibility to generalize the constraints of TAGC is to consider, given an underlying tree automaton with state set Q, monadic first order formulas without function symbols, with atoms built over the state symbols of Q, considered as unary predicates, and equality. These formula are interpreted, given a tree t and a run r of the tree automaton on t, over a structure whose domain is the set of subtrees of t, and such that a predicate $q \in Q$ holds on a subtree s of t if there exists a position $p \in \mathcal{P}os(t)$ such that $t|_p = s$ and $r(p) = q$.

The monadic class of first-order formulas is closely related to set constraints, and decision algorithms for these constraints [Bachmair et al., 1993; Charatonik and Pacholski, 1994] could be a good starting point for studying emptiness decision of tree automata with monadic first order constraints.

TA with Global Monadic Second Order Constraints

We have seen that the emptiness decision procedure of Section I.3.4 also work, on the one hand for an extension of $\mathsf{TAGC}[\approx, \not\approx_{\mathsf{irr}}, \not\approx_{\mathsf{ref}}]$ with local equality and disequality constraints between sibling subtrees, like those of TAB (defined in Section I.1.1), and on the other hand for unranked ordered tree automata with global constraints like those of TAGC.

There exist a decidable model of unranked ordered tree automata with local equality and disequality tests between siblings subtrees called UTASC [Wong and Löding, 2007; Löding and Wong, 2009], generalizing TAB to unranked ordered trees. The local constraints of UTASC use monadic second-order formula with 2 free variables (over words) used to select the pairs of sibling positions of subtrees to be tested for equality or disequality. The decidability of emptiness is open for unranked tree automata combining global constraints like those of TAGC and local constraints like those of UTASC.

More generally, we could study unranked ordered tree automata extended with global constraints using monadic second-order formula (over trees) for the selection of the pairs positions of subtrees to be tested (following *e.g.* [Niehren et al., 2005]). Emptiness is undecidable for such automata, and the problem would be to identify decidable relevant fragments.

Bibliography

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

R. Abbassi, F. Jacquemard, M. Rusinowitch, and S. Guemara El Fatmi. XML access control: from XACML to annotated schemas. In *Intl. Conf. on Communications and Networking (ComNet)*, pages 1–8, 2010. doi: http://dx.doi.org/10.1109/COMNET.2010.5699810. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/AJRG-comnet10.pdf.

P. A. Abdulla, B. Jonsson, P. Mahata, and J. d'Orso. Regular tree model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification*, CAV'02, pages 555–568, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-43997-8. URL http://portal.acm.org/citation.cfm?id=647771.734407.

S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995. ISBN 0-201-53771-0.

S. Abiteboul, O. Benjelloun, and T. Milo. The active xml project: an overview. *The VLDB Journal*, 17:1019–1040, August 2008. ISSN 1066-8888. doi: http://dx.doi.org/10.1007/s00778-007-0049-y. URL http://dx.doi.org/10.1007/s00778-007-0049-y.

S. Abiteboul, P. Bourhis, and B. Marinoiu. Satisfiability and relevance for queries over active documents. In J. Paredaens and J. Su, editors, *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009*, pages 87–96. ACM, 2009. ISBN 978-1-60558-553-6.

R. Affeldt and H. Comon-Lundh. Verification of security protocols with a bounded number of sessions based on resolution for rigid variables. In *Formal to Practical Security*, volume 5458 of *Lecture Notes in Computer Science, State-of-the-Art Survey*, pages 1–20. Springer, May 2009. URL http://staff.aist.go.jp/reynald.affeldt/documents/rigid.pdf.

R. Alur and P. Madhusudan. Visibly pushdown languages. In *36th Annual ACM Symposium on Theory of Computing, STOC*, pages 202–211. ACM, 2004. ISBN 1-58113-852-0.

R. Alur, S. Chaudhuri, and P. Madhusudan. Languages of nested trees. In T. Ball and R. B. Jones, editors, *Proceedings of the 18th International Conference on Computer Aided Verification (CAV)*, volume 4144 of *Lecture Notes in Computer Science*, pages 329–342. Springer, 2006.

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

R. M. Amadio and W. Charatonik. On name generation and set-based analysis in the dolev-yao model. In *13th International Conference on Concurrency Theory, CONCUR*, volume 2421 of *Lecture Notes in Computer Science*, pages 499–514. Springer, 2002. ISBN 3-540-44043-7.

S. Anantharaman, P. Narendran, and M. Rusinowitch. Closure properties and decision problems of dag automata. *Inf. Process. Lett.*, 94(5):231–240, 2005. ISSN 0020-0190. doi: http://dx.doi.org/10.1016/j.ipl.2005.02.004.

P. B. Andrews. Theorem proving via general matings. *Journal of the ACM*, 28(2): 193–214, 1981.

F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, New York, 1998. ISBN 0-521-45520-0.

L. Bachmair. *Canonical Equational Proofs.* Birkhäuser, Boston, 1991.

L. Bachmair and H. Ganzinger. Resolution theorem proving. In J. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter 2. Elsevier, 2001.

L. Bachmair, H. Ganzinger, and U. Waldmann. Set constraints are the monadic class. In *Proceedings of Eighth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 75 – 83. IEEE Computer Society Press, 1993. doi: 10.1109/LICS.1993.287598.

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

A. Barecka and W. Charatonik. The parameterized complexity of chosen problems for finite automata on trees. In *proceedings of the 5th International Conference on Language and Automata Theory and Applications (LATA'09)*, 2011.

L. Barguñó, C. Creus, G. Godoy, F. Jacquemard, and C. Vacher. The Emptiness Problem for Tree Automata with Global Constraints. In J.-P. Jouannaud, editor, *Proceedings of the 25th Annual IEEE Symposium on Logic in*

Computer Science (LICS), pages 263–272, Edinburgh, Scotland, United Kingdom, 2010. IEEE Computer Society Press. doi: 10.1109/LICS.2010.28. URL http://hal.inria.fr/inria-00578901/en.

M. Benedikt and J. Cheney. Semantics, types and effects for xml updates. In *Proceedings 12th International Symposium on Database Programming Languages (DBPL)*, volume 5708 of *Lecture Notes in Computer Science*, pages 1–17, Lyon, France, August 2009. Springer.

V. Benzaken, G. Castagna, and A. Frisch. Cduce: An xml-centric general-purpose language. In *Proceedings of the ACM International Conference on Functional Programming*, 2003.

A. Berlea and H. Seidl. Binary queries for document trees. *Nord. J. Comput.*, 11(1): 41–71, 2004.

B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th IEEE workshop on Computer Security Foundations*, CSFW '01, pages 82–, Washington, DC, USA, 2001. IEEE Computer Society. URL http://portal.acm.org/citation.cfm?id=872752.873511.

A. Blumensath and E. Grädel. Automatic structures. In *In 15th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 51–62. IEEE Computer Society, 2000.

B. Bogaert. *Automates d'arbres avec tests d'égalités*. PhD thesis, LIFL, Université des Science et Technologies de Lille, 1990.

B. Bogaert and S. Tison. Equality and Disequality Constraints on Direct Subterms in Tree Automata. In A. Finkel and M. Jantzen, editors, *9th Symp. on Theoretical Aspects of Computer Science, STACS*, volume 577 of *Lecture Notes in Computer Science*, pages 161–171, Cachan, France, February 13-15 1992. Springer.

B. Bogaert, F. Seynhaeve, and S. Tison. The recognizability problem for tree automata with comparisons between brothers. In W. Thomas, editor, *Foundations of Software Science and Computation Structure, Second International Conference, FoSSaCS'99*, volume 1578 of *Lecture Notes in Computer Science*, pages 150–164. Springer, Mar. 1999. ISBN 3-540-65719-3.

A. Boiret. Grammaires context-free pour les arbres sans rang. Rapport de Master, Master Parisien de Recherche en Informatique, Paris, France, Sept. 2010. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/boiret-m2.pdf.

M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, pages 7–16, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2631-4. doi: 10.1109/LICS.2006.51. URL http://portal.acm.org/citation.cfm?id=1157735.1158037.

M. Bojańczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and xml reasoning. *Journal of the ACM*, 56(3):13:1–13:48, May 2009. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/1516512.1516515. URL http://doi.acm.org/10.1145/1516512.1516515. A preliminary version was presented at PODS'06.

A. Bouajjani and T. Touili. Extrapolating tree transformations. In E. Brinksma and K. Larsen, editors, *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 349–363. Springer Berlin / Heidelberg, 2002. URL http://dx.doi.org/10.1007/3-540-45657-0_46.

A. Bouajjani and T. Touili. On computing reachability sets of process rewrite systems. In J. Giesl, editor, *Proceedings of the 16th International Conference on Term Rewriting and Applications (RTA)*, volume 3467 of *Lecture Notes in Computer Science*, pages 484–499, Nara, Japan, April 19-21 2005. Springer. ISBN 3-540-25596-6.

A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR '97: Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer Berlin / Heidelberg, 1997. URL http://dx.doi.org/10.1007/3-540-63141-0_10.

A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *Proc. of the 12th Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418, 2000.

A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract regular tree model checking of complex dynamic data structures. In K. Yi, editor, *Static Analysis*, volume 4134 of *Lecture Notes in Computer Science*, pages 52–70. Springer Berlin / Heidelberg, 2006. URL http://dx.doi.org/10.1007/11823230_5.

A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *Proceedings of the 16th international symposium on Fundamentals of Computation Theory*, pages 1–22, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74239-5. doi: 10.1007/978-3-540-74240-1_1. URL http://portal.acm.org/citation.cfm?id=1421249.1421251.

A. Bouhoula. Automated theorem proving by test set induction. *Journal of Symbolic Computation*, 23(1):47–77, 1997.

A. Bouhoula and F. Jacquemard. Automating sufficient completeness check for conditional and constrained trs. In *Proceedings of the 20th International Workshop on Unification*, 2006.

A. Bouhoula and F. Jacquemard. Verifying regular trace properties of security protocols with explicit destructors and implicit induction. In *Proc. of the*

Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA'07), pages 27–44, 2007. URL http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2008-07.pdf.

A. Bouhoula and F. Jacquemard. Automated induction with constrained tree automata. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR)*, volume 5195 of *Lecture Notes in Artificial Intelligence*, pages 539–553, Sydney, Australia, Aug. 2008. Springer-Verlag. doi: 10.1007/978-3-540-71070-7_44. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/BJ-ijcar08.pdf.

A. Bouhoula and F. Jacquemard. Sufficient completeness verification for conditional and constrained trs. *Journal of Applied Logic*, 2011. ISSN 1570-8683. doi: 10.1016/j.jal.2011.09.001. URL http://www.sciencedirect.com/science/article/pii/S1570868311000413.

A. Bouhoula and J.-P. Jouannaud. Automata-driven automated induction. *Information and Computation*, 169(1):1–22, 2001.

A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235, 1995.

A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236(1-2):35–132, 2000.

P. Bouyer, A. Petit, and D. Thérien. An algebraic characterization of data and timed languages. In *Proceedings of the 12th International Conference on Concurrency Theory*, CONCUR '01, pages 248–261, London, UK, 2001. Springer-Verlag. ISBN 3-540-42497-0. URL http://portal.acm.org/citation.cfm?id=646736.701768.

W. S. Brainerd. Semi-thue systems and representations of trees. In *Proceedings of the 10th Annual Symposium on Switching and Automata Theory (swat 1969)*, pages 240–244, Washington, DC, USA, 1969. IEEE Computer Society. doi: 10.1109/SWAT.1969.19. URL http://portal.acm.org/citation.cfm?id=1443219.1443424.

L. Bravo, J. Cheney, and I. Fundulaki. Accon: checking consistency of xml write-access control policies. In *EDBT 2008, 11th International Conference on Extending Database Technology, Nantes, France, March 25-29, 2008, Proceedings*, volume 261 of *ACM International Conference Proceeding Series*, pages 715–719. ACM, 2008.

J. A. Bull and D. J. Otway. The authentication protocol. Technical report, Defence Research Agency, Malvern, UK, 1997.

P. Buneman, W. Fan, J. Siméon, and S. Weinstein. Constraints for semistructured data and xml. *SIGMOD Rec.*, 30:47–54, March 2001. ISSN 0163-5808. doi: http://doi.acm.org/10.1145/373626.373697. URL http://doi.acm.org/10.1145/373626.373697.

S. C. Lim and S. H. Son. Access control of xml documents considering update operations. In *Proc. of ACM Workshop on XML Security*, 2003.

L. Cardelli and G. Ghelli. Tql: a query language for semistructured data based on the ambient logic. *Mathematical. Structures in Comp. Sci.*, 14:285–327, June 2004. ISSN 0960-1295. doi: 10.1017/S0960129504004141. URL http://portal.acm.org/citation.cfm?id=992032.992034.

J. Carme, J. Niehren, and M. Tommasi. Querying unranked trees with stepwise tree automata. In *International Conference on Rewriting Techniques and Applications, Aachen*, volume 3091 of *Lecture Notes in Computer Science*, pages 105–118. Springer, 2004.

A.-C. Caron. *Structures et Décision en Réécriture*. PhD thesis, LIFL, Université des Science et Technologies de Lille, Feb. 1993.

A.-C. Caron, H. Comon, J.-L. Coquidé, M. Dauchet, and F. Jacquemard. Pumping, Cleaning and Symbolic Constraints Solving. In S. Abiteboul and E. Shamir, editors, *21st Int. Coll. on Automata, Languages and Programming (ICALP)*, volume 820 of *Lecture Notes in Computer Science*, pages 436–449. Springer, 1994.

D. Caucal. On the regular structure of prefix rewriting. *Theor. Comput. Sci.*, 106:61–86, January 1992. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/0304-3975(92)90278-N. URL http://dx.doi.org/10.1016/0304-3975(92)90278-N.

J. Chabin and P. Réty. Visibly pushdown languages and term rewriting. In *Proceedings 6th International Symposium on Frontiers of Combining Systems (FroCos 2007)*, volume 4720 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 2007. ISBN 978-3-540-74620-1.

W. Charatonik. Automata on dag representations of finite trees. Technical Report Technical Report MPI-I-99-2-001, Max-Planck-Institut fÂ¸r Informatik, SaarbrÂ¸cken, Germany, 1999.

W. Charatonik and L. Pacholski. Negative set constraints with equality. In *Proceedings Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 128–136. IEEE Comput. Soc. Press, 1994. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=316078.

W. Charatonik and L. Pacholski. Set constraints with projections. *J. ACM*, 57: 23:1–23:37, May 2010. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/1734213.1734217. URL http://doi.acm.org/10.1145/1734213.1734217.

W. Charatonik and A. Podelski. Set constraints with intersection. *Information and Computation*, 179(2):213 – 229, 2002. ISSN 0890-5401. doi: DOI:10.1006/inco.2001.2952. URL http://www.sciencedirect.com/science/article/pii/S0890540101929529.

M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. *Maude: Specification and Programming in Rewriting Logic*. SRI International, 1999.

T. Colcombet. Rewriting in the partial algebra of typed terms modulo ac. *Electr. Notes Theor. Comput. Sci.*, 68(6), 2002.

H. Comon. Sequentiality, second order monadic logic and tree automata. In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 508–517. IEEE Computer Society, 1995.

H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. *Theoretical Computer Science*, 331(1):143–214, Feb. 2005. doi: 10.1016/j.tcs.2004.09.036. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PS/ComonCortierTCS1.ps.

H. Comon and F. Jacquemard. Ground reducibility is EXPTIME-complete. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 26–34, Warsaw, Poland, 27-30 july 1997. IEEE Computer Society Press.

H. Comon and F. Jacquemard. Ground reducibility is EXPTIME-complete. *Information and Computation*, 187(1):123–153, Nov. 2003.

H. Comon, M. Haberstrau, and J. P. Jouannaud. Syntacticness, cycle-syntacticness, and shallow theories. *Information and Computation*, 111(1): 154 – 191, 1994. ISSN 0890-5401. doi: DOI:10.1006/inco.1994.1043. URL http://www.sciencedirect.com/science/article/pii/S0890540184710431.

H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, C. Löding, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. http://tata.gforge.inria.fr, 2007.

H. Comon-Lundh. Inductionless induction. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 14, pages 913–962. Elsevier, 2001.

H. Comon-Lundh, F. Jacquemard, and N. Perrin. Tree automata with memory, visibility and structural constraints. In H. Seidl, editor, *Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'07)*, volume 4423 of *Lecture Notes in Computer Science*, pages 168–182. Springer, Mar. 2007. doi: 10.1007/978-3-540-71389-0_13. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/CJP-fossacs07.pdf.

H. Comon-Lundh, F. Jacquemard, and N. Perrin. Visibly tree automata with memory and constraints. *Logical Methods in Computer Science*, 4(2:8), 2008. doi: 10.2168/LMCS-4(2:8)2008.

J.-L. Coquidé, M. Dauchet, R. Gilleron, and S. Vágvölgyi. Bottom-up tree push-down automata: classification and connection with rewrite systems. *Theoretical Computer Science*, 127(1):69–98, 1994.

B. Courcelle. On recognizable sets and tree automata. In M. Nivat and H. Ait Kaci, editors, *Resolution of Equations in Algebraic Structures*, volume 1, Algebraic techniques, pages 93–126. Academic Press, 1989. ISBN 0-12-046370-9.

B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 2, chapter 5, pages 193–242. Elsevier, 1990.

S. Dal Zilio and D. Lugiez. XML schema, tree logic and sheaves automata. *Journal Applicable Algebra in Engineering, Communication and Computing*, 17(5):337–377, 2006.

E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Securing xml documents. In *Proceedings of the 7th International Conference on Extending Database Technology (EDBT 2000)*, volume 1777 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2000. ISBN 3-540-67227-3.

J. Dassow, G. Paun, and A. Salomaa. Grammars with controlled derivations. In *Handbook of Formal Languages*, volume 2, chapter 3, pages 101–154. Springer, 1997. ISBN 3-540-60648-3.

M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proceedings Fifth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 242–248. IEEE Computer Society, 1990.

M. Dauchet, S. Tison, T. Heuillard, and P. Lescanne. Decidability of the confluence of ground term rewriting systems. In *Proceedings of the second Symposium on Logic in Computer Science (LICS)*, pages 353–359. The Computer Society of the IEEE, 22–25 jun 1987. ISBN 0-8186-0793-9.

M. Dauchet, A.-C. Caron, and J.-L. Coquidé. Reduction properties and automata with constraints. *Journal of Symbolic Computation*, 20(2):215–233, 1995.

C. David, L. Libkin, and T. Tan. Efficient reasoning about data trees via integer linear programming. In *Proceedings of the 14th International Conference on Database Theory (ICDT)*, ICDT '11, pages 18–29, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0529-7. doi: http://doi.acm.org/10.1145/1938551.1938558. URL http://doi.acm.org/10.1145/1938551.1938558.

S. Delaune and F. Jacquemard. A decision procedure for the verification of security protocols with explicit destructors. In *Proceedings of the 11th ACM conference on Computer and communications security*, CCS '04, pages 278–287, New York, NY, USA, 2004. ACM. ISBN 1-58113-961-6. doi: http://doi.acm.org/10.1145/1030083.1030121. URL http://doi.acm.org/10.1145/1030083.1030121.

G. Delzanno, J.-F. Raskin, and L. V. Begin. Towards the automated verification of multithreaded java programs. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS '02, pages 173–187, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-43419-4. URL http://portal.acm.org/citation.cfm?id=646486.694620.

S. Demri and R. Lazić. Ltl with the freeze quantifier and register automata. *ACM Trans. Comput. Logic*, 10:16:1–16:30, April 2009. ISSN 1529-3785. doi: http://doi.acm.org/10.1145/1507244.1507246. URL http://doi.acm.org/10.1145/1507244.1507246.

N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume volume B: Formal Models and Semantics, chapter 6, pages 243–320. North-Holland, Amsterdam, 1990.

J. d'Orso and T. Touili. Regular hedge model checking. In *In Proc. of the 4th IFIP International Conference on Theoretical Computer Science (TCS'06)*. IFIP, 2006.

I. Durand and A. Middeldorp. Decidable call-by-need computations in term rewriting. *Information and Computation*, 196(2):95 – 126, 2005. ISSN 0890-5401. doi: DOI:10.1016/j.ic.2004.10.003. URL http://www.sciencedirect.com/science/article/B6WGK-4F7VFHM-2/2/d7b66e9e8ca0d695ab486fb0

I. Durand and G. Sénizergues. Bottom-up rewriting is inverse recognizability preserving. In F. Baader, editor, *Proceedings of the 18th International Conference on Term Rewriting and Applications (RTA)*, volume 4533 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2007.

I. Durand and M. Sylvestre. Left-linear Bounded TRSs are Inverse Recognizability Preserving. In M. Schmidt-Schauß, editor, *22nd International Conference on Rewriting Techniques and Applications (RTA'11)*, volume 10 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 361–376, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-30-9. doi: http://dx.doi.org/10.4230/LIPIcs.RTA.2011.361. URL http://drops.dagstuhl.de/opus/volltexte/2011/3135.

I. Durand, G. Sénizergues, and M. Sylvestre. Termination of linear bounded term rewriting systems. In C. Lynch, editor, *Proceedings of the 21st International Conference on Rewriting Techniques and Applications*, volume 6 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 341–356, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-18-7. doi: http://dx.doi.org/10.4230/LIPIcs.RTA.2010.341. URL http://drops.dagstuhl.de/opus/volltexte/2010/2662.

J. Engelfriet. Top-down tree transducers with regular look-ahead. *Theory of Computing Systems*, 10:289–303, 1976. ISSN 1432-4350. URL http://dx.doi.org/10.1007/BF01683280.

J. Engelfriet. Three hierarchies of transducers. *Theory of Computing Systems*, 15: 95–125, 1981. ISSN 1432-4350. URL http://dx.doi.org/10.1007/BF01786975. 10.1007/BF01786975.

J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comp. Syst. Sci.*, 31:71–146, 1985.

J. Engelfriet, S. Maneth, and H. Seidl. Deciding equivalence of top-down xml transformations in polynomial time. *J. Comput. Syst. Sci.*, 75(5):271–286, 2009.

R. Fagin, B. Kimelfeld, Y. Li, S. Raghavan, and S. Vaithyanathan. Rewrite rules for search database systems. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '11, pages 271–282, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0660-7. doi: http://doi.acm.org/10.1145/1989284.1989322. URL http://doi.acm.org/10.1145/1989284.1989322.

W. Fan and L. Libkin. On xml integrity constraints in the presence of dtds. *J. ACM*, 49:368–406, May 2002. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/567112.567117. URL http://doi.acm.org/10.1145/567112.567117.

W. Fan and J. Siméon. Integrity constraints for xml. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '00, pages 23–34, New York, NY, USA, 2000. ACM. ISBN 1-58113-214-X. doi: http://doi.acm.org/10.1145/335168.335172. URL http://doi.acm.org/10.1145/335168.335172.

W. Fan, C.-Y. Chan, and M. Garofalakis. Secure xml querying with security views. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data (SIGMOD'04)*, pages 587–598, New York, NY, USA, 2004. ACM. ISBN 1-58113-859-8. doi: http://doi.acm.org/10.1145/1007568.1007634.

C. G. Fermüller, A. Leitsch, U. Hustadt, and T. Tammet. *Resolution decision procedures*, pages 1791–1849. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, 2001. ISBN 0-444-50812-0. URL http://portal.acm.org/citation.cfm?id=778522.778534.

E. Filiot, J.-M. Talbot, and S. Tison. Satisfiability of a spatial logic with tree variables. In *Proceedings of the 21st International Workshop on Computer Science Logic (CSL 2007)*, volume 4646 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2007. ISBN 978-3-540-74914-1.

E. Filiot, J.-M. Talbot, and S. Tison. Tree automata with global constraints. In *12th International Conference in Developments in Language Theory (DLT 2008)*, volume 5257 of *Lecture Notes in Computer Science*, pages 314–326. Springer, 2008. ISBN 978-3-540-85779-2.

N. Francis, C. David, and L. Libkin. A direct translation from xpath to nondeterministic automata. In *5th Alberto Mendelzon International Workshop on Foundations of Data Management*, 2011.

L. Fribourg. A strong restriction of the inductive completion procedure. *Journal of Symbolic Computation*, 8(3):253–276, 1989.

M. Frick, M. Grohe, and C. Koch. Query evaluation on compressed trees (extended abstract). In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, pages 188–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1884-2. URL http://portal.acm.org/citation.cfm?id=788023.789040.

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

I. Fundulaki and S. Maneth. Formalizing xml access control for update operations. In *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 169–174, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-745-2. doi: http://doi.acm.org/10.1145/1266840.1266868.

K. Futatsugi, J. A. Goguen, J.-P. Jouannaud, and J. Meseguer. Principles of obj2. In *Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '85, pages 52–66, New York, NY, USA, 1985. ACM. ISBN 0-89791-147-4. doi: http://doi.acm.org/10.1145/318593.318610. URL http://doi.acm.org/10.1145/318593.318610.

H. Ganzinger and J. Stuber. Inductive theorem proving by consistency for first-order clauses. In *Proceedings of the Third International Workshop on Conditional Term Rewriting Systems*, pages 226–241, London, UK, 1993. Springer-Verlag. ISBN 3-540-56393-8. URL http://portal.acm.org/citation.cfm?id=648339.756052.

H. Ganzinger, F. Jacquemard, and M. Veanes. Rigid reachability. In *Proc. ASIAN'98*, volume 1538 of *Lecture Notes in Computer Science*, pages 4–??, Berlin, 1998. Springer-Verlag.

H. Ganzinger, F. Jacquemard, and M. Veanes. Rigid reachability: The non-symmetric form of rigid E-unification. *Intl. Journal of Foundations of Computer Science*, 11 (1):3–27, 2000.

A. Gascón, G. Godoy, and F. Jacquemard. Closure of tree automata languages under innermost rewriting. In A. Middeldorp, editor, *Proceedings of the 8th International Workshop on Reduction Strategies in Rewriting and Programming (WRS'08)*, volume 237 of *Electronic Notes in Theoretical Computer Science*, pages 23–38, Castle of Hagenberg, Austria, July 2008. Elsevier Science Publishers. URL http://hal.inria.fr/inria-00578966/en.

T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Proc. of 17th Int. Conf. on Automated Deduction, CADE*, volume 1831 of *Lecture Notes in Computer Science*, pages 271–290. Springer, 2000.

T. Genet and V. Rusu. Equational approximations for tree automata completion. *Journal of Symbolic Computation*, 45(5):574 – 597, 2010. ISSN 0747-7171. doi: DOI:10.1016/j.jsc.2010.01.009. URL http://www.sciencedirect.com/science/article/B6WM7-4Y95RK3-3/2/2124c281aa8809d67b302fd Symbolic Computation in Software Science.

T. Genet and V. V. T. Tong. Reachability analysis of term rewriting systems with timbuk. In *Proceedings of the Artificial Intelligence on Logic for Programming*, LPAR '01, pages 695–706, London, UK, 2001. Springer-Verlag. ISBN 3-540-42957-3. URL http://portal.acm.org/citation.cfm?id=645710.664452.

A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On tree automata that certify termination of left-linear term rewriting systems. *Information and Computation*, 205(4):512 – 534, 2007. ISSN 0890-5401. doi: DOI:10.1016/j.ic.2006.08.007. URL http://www.sciencedirect.com/science/article/pii/S0890540106001544. Special Issue: 16th International Conference on Rewriting Techniques and Applications.

R. Gilleron. Decision problems for term rewriting systems and recognizable tree languages. In *Proceedings of the 8th annual symposium on Theoretical aspects of computer science (STACS)*, volume 480 of *Lecture Notes in Computer Science*, pages 148–159. Springer-Verlag New York, Inc., 1991. ISBN 0-387-53709-0.

R. Gilleron and S. Tison. Regular tree languages and rewrite systems. *Fundam. Inform.*, 24(1/2):157–176, 1995.

G. Godoy and H. Hernandez. Undecidable properties of flat term rewrite systems. *Appl. Algebra Eng., Commun. Comput.*, 20:187–205, June 2009. ISSN 0938-1279. doi: 10.1007/s00200-009-0097-1. URL http://portal.acm.org/citation.cfm?id=1554984.1554989.

G. Godoy and E. Huntingford. Innermost-reachability and innermost-joinability are decidable for shallow term rewrite systems. In F. Baader, editor, *Proceedings of the 18th International Conference of Rewriting Techniques and applications (RTA)*, volume 4533 of *Lecture Notes in Computer Science*, pages 184–199. Springer, July 2007.

G. Godoy and F. Jacquemard. Unique normalization for shallow trs. In R. Treinen, editor, *20th International Conference on Rewriting Techniques and Applications (RTA)*, volume 5595 of *Lecture Notes in Computer Science*, pages 63–77, Brazilia, Brésil, 2009. Springer. doi: 10.1007/978-3-642-02348-4_5. URL http://hal.inria.fr/inria-00578959/en/.

G. Godoy and S. Tison. On the normalization and unique normalization properties of term rewrite systems. In F. Pfenning, editor, *Proc. 21st International Conference on Automated Deduction (CADE)*, volume 4603 of *Lecture Notes in Computer Science*, pages 247–262, July 2007.

G. Godoy and A. Tiwari. Deciding fundamental properties of right-(ground or variable) rewrite systems by rewrite closure. In D. Basin and M. Rusinowitch, editors, *Intl. Joint Conf. on Automated Deduction, IJCAR*, volume 3097 of *LNAI*, pages 91–106. Springer, July 2004.

G. Godoy and A. Tiwari. Confluence of shallow right-linear rewrite systems. In C.-H. L. Ong, editor, *19th International Workshop of Computer Science Logic, CSL*, volume 3634 of *Lecture Notes in Computer Science*, pages 541–556. Springer, Aug. 2005.

G. Godoy, E. Huntingford, and A.Tiwari. Termination of rewriting with right-flat rules. In F. Baader, editor, *18th International Conference on Term Rewriting and Applications (RTA)*, volume 4533 of *Lecture Notes in Computer Science*, pages 200–213. Springer, June 2007.

G. Godoy, O. Giménez, L. Ramos, and C. Àlvarez. The hom problem is decidable. In L. J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 485–494, Cambridge, Massachusetts, USA, 5-8 June 2010.

G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. *Journal of the ACM*, 51(1):74–113, 2004.

J. Goubault-Larrecq. Deciding $\mathcal{H}_1$ by resolution. *Information Processing Letters*, 95(3):401–408, Aug. 2005. doi: 10.1016/j.ipl.2005.04.007. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/Goubault-h1.pdf.

J. Goubault-Larrecq. Démonstration automatique. MPRI Lecture Notes, 2006. URL http://www.lsv.ens-cachan.fr/~goubault/DemAuto/demauto.html.

J. Goubault-Larrecq, M. Roger, and K. N. Verma. Abstraction and resolution modulo ac: How to verify diffie-hellman-like protocols automatically. *Journal of Logic and Algebraic Programming*, 64(2):219–251, 2005. URL http://www2.in.tum.de/~verma/publications/GLRV-acm.pdf.

I. Guessarian. Pushdown tree automata. *Mathematical Systems Theory*, 16(1):237–263, 1983.

P. Gyenizse and S. Vágvölgyi. Linear generalized semi-monadic rewrite systems effectively preserve recognizability. *Theoretical Computer Science*, 194:87–122, March 1998. ISSN 0304-3975. doi: 10.1016/S0304-3975(96)00333-7. URL http://portal.acm.org/citation.cfm?id=271246.271255.

P. Habermehl, R. Iosif, and T. Vojnar. Automata-based verification of programs with tree updates. *Acta Informatica*, 47:1–31, 2010. ISSN 0001-5903. URL http://dx.doi.org/10.1007/s00236-009-0108-5. 10.1007/s00236-009-0108-5.

D. Hofbauer and J. Waldmann. Deleting string rewriting systems preserve regularity. *Theor. Comput. Sci.*, 327(3):301–317, 2004. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/j.tcs.2004.04.009.

J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley series in computer science. Addison-Wesley, Reading, Mass., 1979. ISBN 020102988X.

H. Hosoya, J. Vouillon, and B. C. Pierce. Regular expression types for xml. *ACM Trans. Program. Lang. Syst.*, 27:46–90, January 2005. ISSN 0164-0925. doi: http://doi.acm.org/10.1145/1053468.1053470. URL http://doi.acm.org/10.1145/1053468.1053470.

G. P. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *J. Comput. Syst. Sci.*, 25(2):239–266, 1982.

F. Jacquemard. Decidable approximations of term rewriting systems. In H. Ganzinger, editor, *Proceedings of the 7th International Conference on Rewriting Techniques and Applications (RTA)*, volume 1103 of *Lecture Notes in Computer Science*, pages 362–376. Springer, 1996.

F. Jacquemard. Reachability and confluence are undecidable for flat term rewriting systems. *Inf. Process. Lett.*, 87(5):265–270, 2003.

F. Jacquemard and M. Rusinowitch. Closure of Hedge-automata languages by Hedge rewriting. In A. Voronkov, editor, *Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA'08)*, volume 5117 of *Lecture Notes in Computer Science*, pages 157–171, Hagenberg, Austria, July 2008a. Springer. doi: 10.1007/978-3-540-70590-1_11. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/JR-rta08.pdf.

F. Jacquemard and M. Rusinowitch. Rewrite closure of Hedge-automata languages. Research Report LSV-08-05, Laboratoire Spécification et Vérification, ENS Cachan, France, 2008b. URL http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2008-05.pdf.

F. Jacquemard and M. Rusinowitch. Rewrite-based verification of xml updates. In *Proceedings of the 12th ACM SIGPLAN International Symposium on Principles and Practice of Declarative Programming (PPDP)*, PPDP '10, pages 119–130, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0132-9. doi: http://doi.acm.org/10.1145/1836089.1836105.

F. Jacquemard, C. Meyer, and C. Weidenbach. Unification in Extensions of Shallow Equational Theories. In *9th Int. Conf. on Rewriting Techniques and Applications, RTA*, volume 1379 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 1998.

F. Jacquemard, M. Rusinowitch, and L. Vigneron. Tree automata with equality constraints modulo equational theories. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 557–571, Seattle, Washington, USA, Aug. 2006. Springer-Verlag. doi: 10.1007/11814771_45.

F. Jacquemard, M. Rusinowitch, and L. Vigneron. Tree automata with equality constraints modulo equational theories. *Journal of Logic and Algebraic Programming*, 75(2):182–208, Apr. 2008a. doi: 10.1016/j.jlap.2007.10.006. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/JRV-jlap08.pdf.

F. Jacquemard, M. Rusinowitch, and L. Vigneron. Tace: Library of tree automata with constraints, 2008b. URL http://tace.gforge.inria.fr.

F. Jacquemard, F. Klay, and C. Vacher. Rigid tree automata. In A. Horia Dediu, A. Mihai Ionescu, and C. Martín-Vide, editors, *Proceedings of the 3rd International Conference on Language and Automata Theory and Applications (LATA'09)*, volume 5457 of *Lecture Notes in Computer Science*, pages 446–457, Tarragona, Spain, Apr. 2009. Springer.

F. Jacquemard, F. Klay, and C. Vacher. Rigid tree automata and applications. *Inf. Comput.*, 209(3):486–512, 2011a.

F. Jacquemard, Y. Kojima, and M. Sakai. Controlled term rewriting. In C. Tinelli and V. Sofronie-Stokkermans, editors, *Proceedings of the 8th International Symposium Frontiers of Combining Systems (FroCoS)*, volume 6989 of *Lecture Notes in Artificial Intelligence*, pages 179–194, Heidelberg, 2011b. Springer.

F. Jacquemard, E. Lozes, R. Treinen, and J. Villard. Multiple congruence relations, first-order theories on terms, and the frames of the applied pi-calculus. In *Theory of Security and Applications (TOSCA)*, Lecture Notes in Computer Science, Saarbrücken, Allemagne, 2011c. Springer. URL http://hal.inria.fr/inria-00578896/en/. To appear.

N. D. Jones and N. Andersen. Flow analysis of lazy higher-order functional programs. *Theoretical Computer Science*, 375(1-3):120 – 136, 2007. ISSN 0304-3975. doi: 10.1016/j.tcs.2006.12.030. URL http://www.sciencedirect.com/science/article/B6V1G-4MPC46J-5/2/9270a15e14345e4de06b194

J.-P. Jouannaud and E. Kounalis. Proof by induction in equational theories without constructors. In *In Proc. 1st IEEE Symposium on Logic in Computer Science*, 1986.

M. Jurdzinski and R. Lazic. Alternation-free modal mu-calculus for data trees. In *Proceedings 22nd IEEE Symposium on Logic in Computer Science (LICS)*, pages 131–140. IEEE Computer Society, 2007.

D. Kapur and D. R. Musser. Proof by consistency. *The Artificial Intelligence Journal*, 31:125–157, 1987.

M. Kay. Xsl transformations (xslt) 2.0. W3c working draft, World Wide Web Consortium, 2003. Available at http://www.w3.org/TR/xslt20.

C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on Automatic Deduction.

F. Klaedtke and H. Ruess. Parikh automata and monadic second-order logics with linear cardinality constraints. Technical Report 177, Intitute of Computer Science at Freiburg University, 2002.

N. Klarlund and A. Møller. *MONA Version 1.4 User Manual*. BRICS, Department of Computer Science, Aarhus University, January 2001. Notes Series NS-01-1. Available from `http://www.brics.dk/mona/`. Revision of BRICS NS-98-3.

C. Koch. Efficient processing of expressive node-selecting queries on xml data in secondary storage: a tree automata-based approach. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '2003, pages 249–260. VLDB Endowment, 2003. ISBN 0-12-722442-4. URL http://portal.acm.org/citation.cfm?id=1315451.1315474.

J. Kochems and L. Ong. Improved functional flow and reachability analyses using indexed linear tree grammars. In M. Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications (RTA)*, volume 10 of *LIPIcs*, pages 187–202, Novi Sad, Serbia, 2011. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Y. Kojima and M. Sakai. Innermost reachability and context sensitive reachability properties are decidable for linear right-shallow term rewriting systems. In A. Voronkov, editor, *Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA)*, volume 5117 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2008. ISBN 978-3-540-70588-8.

G. Kucherov and M. Tajine. Decidability of regularity and related properties of ground normal form languages. *Information and Computation*, 118(1):91–100, April 1995.

S. Y. Kuroda. Classes of languages and linear-bounded automata. *Information and Control*, 7:207–223, 1964.

C. Löding. Ground tree rewriting graphs of bounded tree width. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS'02)*, volume 2285 of *Lecture Notes in Computer Science*, pages 559–570. Springer, 2002.

C. Löding and A. Spelten. Transition graphs of rewriting systems over unranked trees. In L. Kucera and A. Kucera, editors, *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 67–77, 2007. ISBN 978-3-540-74455-9.

C. Löding and K. Wong. On nondeterministic unranked tree automata with sibling constraints. In R. Kannan and K. N. Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 4 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 311–322. Schloss Dagstuhl - Leibniz-Zentrum für Informatik., 2009.

M. Lohrey and S. Maneth. The complexity of tree automata and xpath on grammar-compressed trees. *Theoretical Computer Science*, 363:196–210, October 2006. ISSN 0304-3975. doi: 10.1016/j.tcs.2006.07.024. URL http://portal.acm.org/citation.cfm?id=1217607.1217615.

S. Lucas. Context-sensitive rewriting strategies. *Inf. Comput.*, 178:294–343, October 2002. ISSN 0890-5401. doi: 10.1016/S0890-5401(02)93176-7. URL http://portal.acm.org/citation.cfm?id=603645.603659.

D. Lugiez. Multitree automata that count. *Theor. Comput. Sci.*, 333:225–263, March 2005. ISSN 0304-3975. doi: 10.1016/j.tcs.2004.10.023. URL http://portal.acm.org/citation.cfm?id=1196006.1196014.

D. Lugiez. Forward analysis of dynamic network of pushdown systems is easier without order. In *Proceedings of the 3rd International Workshop on Reachability Problems*, RP '09, pages 127–140, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-04419-9. doi: http://dx.doi.org/10.1007/978-3-642-04420-5_13. URL http://dx.doi.org/10.1007/978-3-642-04420-5_13.

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

S. Maneth, A. Berlea, T. Perst, and H. Seidl. Xml type checking with macro tree transducers. In *24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, pages 283–294, 2005.

S. Maneth, T. Perst, and H. Seidl. Exact xml type checking in polynomial time. In T. Schwentick and D. Suciu, editors, *Proceedings of the 11th International Conference on Database Theory (ICDT 2007)*, volume 4353 of *Lecture Notes in Computer Science*, pages 254–268. Springer, 2007. ISBN 3-540-69269-X.

W. Martens and F. Neven. Frontiers of tractability for typechecking simple xml transformations. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 23–34. ACM, 2004.

M. Marx. Xpath with conditional axis relations. In E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm, and E. Ferrari, editors, *Advances in Database Technology - EDBT 2004*, volume 2992 of *Lecture Notes in Computer Science*, pages 579–580. Springer Berlin / Heidelberg, 2004. URL http://dx.doi.org/10.1007/978-3-540-24741-8_28.

T. Milo, D. Suciu, and V. Vianu. Typechecking for xml transformers. *J. of Comp. Syst. Sci.*, 66(1):66–97, 2003.

I. Mitsuhashi, M. Oyamaguchi, and F. Jacquemard. The confluence problem for flat TRSs. In *Proc. 8th Intl. Conf. on Artificial Intelligence and Symbolic Computation (AISC'06)*, volume 4120 of *LNAI*, pages 68–81. Springer, 2006.

J. Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG*. PhD thesis, LIFL, Université des Science et Technologies de Lille, 1981.

N. Moore. Computational complexity of the problem of tree generation under fine-grained access control policies. *Inf. Comput.*, 209(3):548–567, 2011.

M. Murata. Hedge automata: a formal model for XML schemata. Technical report, Fuji Xerox INformation Systems, 1999.

M. Murata. Hedge automata: a formal model for XML schemata. Web page, 2000. URL citeseer.nj.nec.com/murata99hedge.html.

M. Murata, D. Lee, and M. Mani. Taxonomy of xml schema languages using formal language theory. In *In Extreme Markup Languages*, 2000.

M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of xml schema languages using formal language theory. *ACM Trans. Internet Technol.*, 5:660–704, November 2005. ISSN 1533-5399. doi: http://doi.acm.org/10.1145/1111627.1111631. URL http://doi.acm.org/10.1145/1111627.1111631.

M. Murata, A. Tozawa, M. Kudo, and S. Hada. Xml access control using static analysis. *ACM Trans. Inf. Syst. Secur.*, 9(3):292–324, 2006. ISSN 1094-9224. doi: http://doi.acm.org/10.1145/1178618.1178621.

T. Nagaya and Y. Toyama. Decidability for left-linear growing term rewriting systems. *Inf. Comput.*, 178(2):499–514, 2002.

A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In V. Arvind and R. Ramanujam, editors, *Proceedings of the 18th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 1530 of *Lecture Notes in Computer Science*, pages 134–145, Chennai, India, December 17-19 1998. Springer. ISBN 3-540-65384-8.

F. Neven and T. Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1-2):633–674, 2002.

F. Neven and J. Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. *Journal of the ACM*, 49(1):56–100, 2002.

F. Neven, T. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science*, MFCS '01, pages 560–572, London, UK, 2001. Springer-Verlag. ISBN 3-540-42496-2. URL http://portal.acm.org/citation.cfm?id=645730.668039.

J. Niehren, L. Planque, J.-M. Talbot, and S. Tison. N-ary queries by tree automata. In *Proceedings of the 10th International Symposium on Database Programming Languages (DBPL)*, volume 3774 of *Lecture Notes in Computer Science*, pages 217–231. Springer, 2005.

F. Nielson, H. R. Nielson, and H. Seidl. Cryptographic Analysis in Cubic Time. *Electr. Notes Theor. Comput. Sci.*, 62, 2001.

F. Nielson, H. Riis Nielson, and H. Seidl. Normalizable Horn Clauses, Strongly Recognizable Relations, and Spi. In *Static Analysis, 9th Int. Symp., SAS*, volume 2477 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2002.

R. Nieuwenhuis and A. Rubio. *Paramodulation-Based Theorem Proving*, chapter Handbook of Automated Reasoning, Volume I, Chapter 7, pages 371–443. Elsevier Science and MIT Press, 2001.

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

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

H. Ohsaki and T. Takai. Decidability and closure properties of equational tree languages. In *13th Int. Conf. on Rewriting Techniques and Applications (RTA)*, volume 2378 of *Lecture Notes in Computer Science*, pages 114–128. Springer, 2002b.

H. Ohsaki, H. Seki, and T. Takai. Recognizing boolean closed a-tree languages with membership conditional rewriting mechanism. In *Proc. of the 14th Int. Conference on Rewriting Techniques and Applications (RTA 2003)*, volume 2706 of *Lecture Notes in Computer Science*, pages 483–498. Springer Verlag, 2003. ISBN 3-540-40254-3.

H. Ohsaki, J.-M. Talbot, S. Tison, and Y. Roos. Monotone AC-tree automata. In *12th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science. Springer, 2005. URL http://www.lifl.fr/~yroos/bib/Monotone-AC-Tree-Automata.pdf.

M. Penttonen. One-sided and two-sided context in formal grammars. *Information and Control*, 25:371–392, 1974.

T. Perst and H. Seidl. Macro forest transducers. *Information Processing Letters*, 89: 141–149, 2004.

Th. Place. *Decidable Characterizations for Tree Logics*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, Dec. 2010. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/place-phd.pdf.

P. Réty and J. Vuotto. Tree automata for rewrite strategies. *J. Symb. Comput.*, 40: 749–794, July 2005. ISSN 0747-7171. doi: http://dx.doi.org/10.1016/j.jsc.2004.12.008. URL http://dx.doi.org/10.1016/j.jsc.2004.12.008.

A. Reußand H. Seidl. Bottom-up tree automata with term constraints. In *Proceedings of the 17th international conference on Logic for programming, artificial intelligence, and reasoning*, LPAR'10, pages 581–593, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16241-X, 978-3-642-16241-1. URL http://portal.acm.org/citation.cfm?id=1928380.1928421.

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

J. Robie, D. Chamberlin, M. Dyck, D. Florescu, J. Melton, and J. Siméon. Xquery update facility 1.0. Technical report, W3C, March 2011. URL http://www.w3.org/TR/xquery-update-10/.

K. Salomaa. Deterministic tree pushdown automata and monadic tree rewriting systems. *J. Comput. Syst. Sci.*, 37(3):367–394, 1988.

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

T. Schwentick. Automata for xml - a survey. *J. Comput. Syst. Sci.*, 73(3):289–315, 2007.

S. Schwoon. *Model-Checking Pushdown Systems*. PhD thesis, Technische Universität München, 2002.

H. Seidl. Haskell overloading is dexptime-complete. *Information Processing Letters*, 52(2):57–60, 1994. ISSN 0020-0190. doi: http://dx.doi.org/10.1016/0020-0190(94)00130-8.

H. Seidl. Program Analysis through Finite Tree Automata. In S. Maneth, editor, *Implementation and Application of Automata*, volume 5642 of *Lecture Notes in Computer Science*, page 3, Sydney, Australia, July 2009. Springer.

H. Seidl and K. N. Verma. Cryptographic Protocol Verification Using Tractable Classes of Horn Clauses. In T. W. Reps, M. Sagiv, and J. Bauer, editors, *Program Analysis and Compilation, Theory and Practice*, volume 4444 of *Lecture Notes in Computer Science*, pages 97–119. Springer, 2007.

H. Seidl and K. N. Verma. Flat and one-variable clauses: Complexity of verifying cryptographic protocols with single blind copying. *ACM Trans. Comput. Log.*, 9 (4), 2008.

H. Seidl and K. N. Verma. Flat and one-variable clauses for single blind copying protocols. In R. Treinen, editor, *Proceedings 20th International Conference on Rewriting Techniques and Applications (RTA)*, volume 5595 of *Lecture Notes in Computer Science*, pages 118–132. Springer, 2009. ISBN 978-3-642-02347-7.

H. Seidl, T. Schwentick, and A. Muscholl. Numerical document queries. In *22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 155–166. ACM Press, 2003.

H. Seidl, T. Schwentick, A. Muscholl, and P. Habermehl. Counting in trees for free. In *In Proc. ICALP'04*, volume 3142 of *LNCS*, pages 1136–1149. Springer-Verlag, 2004.

H. Seidl, T. Schwentick, and A. Muscholl. Counting in trees. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 575–612. Amsterdam University Press, 2008. ISBN 978-90-5356-576-6.

H. Seki, T. Takai, Y. Fujinaka, and Y. Kaji. Layered Transducing Term Rewriting System and Its Recognizability Preserving Property. In *Int. Conf. on Rewriting Techniques and Applications (RTA)*, volume 2378 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2002.

G. Sénizergues. Formal languages and word-rewriting. In H. Comon and J.-P. Jouannaud, editors, *Term Rewriting, Advanced Course of French Spring School of Theoretical Computer Science, Font Romeux, France*, volume 909 of *Lecture Notes in Computer Science*, pages 75–94. Springer, 1993.

T. Takai, Y. Kaji, and H. Seki. Right-linear finite path overlapping term rewriting systems effectively preserve recognizability. In *11th International Conference on Rewriting Techniques and Applications (RTA)*, volume 1833 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 2000.

J. Thatcher and J. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. Syst. Theory*, 2(1):57–81, 1968.

W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 7, pages 389–455. Springer Verlag, 1997.

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

M. Tommasi. Automates d'arbres avec tests d'égalité entre cousins germains. Mémoire de DEA, Univ. Lille I, 1992.

T. Touili. Computing transitive closures of hedge transformations. In *In Proc. 1st International Workshop on Verification and Evaluation of Computer and Communication Systems (VECOS'07)*, eWIC Series. British Computer Society, 2007.

T. Truderung. Selecting theories and recursive protocols. In *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR)*, volume 3653 of *Lecture Notes in Computer Science*, pages 217–232. Springer, 2005.

C. Vacher. *Automates à contraintes globales pour la vérification de propriétés de sécurité*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, Dec. 2010. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/vacher-phd.pdf.

S. Vágvölgyi and R. Gilleron. For a rewrite system it is decidable whether the set of irreducible ground terms is recognizable. *Bulletin of the EATCS*, 48:197–209, 1992. URL http://hal.inria.fr/inria-00538877/en/.

M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. *Logic in Computer Science*, pages 332–344, 1986. URL http://www.cs.utep.edu/sroach/F07-5383/AutomataTheoretic-Approach-SPIN-lics86.pdf.

M. Veanes. On computational complexity of basic decision problems of finite tree automata. Technical report, Uppsala Computing Science Department, 1997.

K. N. Verma. *Two-Way Equational Tree Automata*. PhD thesis, ENS Cachan, Sept. 2003.

K. N. Verma and J. Goubault-Larrecq. Alternating two-way AC-tree automata. *Information and Computation*, 205(6): 817–869, June 2007a. doi: 10.1016/j.ic.2006.12.006. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/VG-icomp07.pdf.

K. N. Verma and J. Goubault-Larrecq. Alternating two-way AC-tree automata. *Information and Computation*, 205(6): 817–869, June 2007b. doi: 10.1016/j.ic.2006.12.006. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/VG-icomp07.pdf.

K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational horn clauses. In *20th Int. Conf. on Automated Deduction (CADE)*, volume 3632 of *LNCS*, pages 337–352. Springer, 2005.

R. Verma. Complexity of normal form properties and reductions for rewriting problems. *Fundamenta Informaticae*, 2008. Accepted for publication.

R. M. Verma, M. Rusinowitch, and D. Lugiez. Algorithms and reductions for rewriting problems. *Fundam. Inf.*, 46:257–276, August 2001. ISSN 0169-2968. URL http://portal.acm.org/citation.cfm?id=1219995.1220000.

V. Vianu. A web odyssey: from codd to xml. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, pages 1–15, New York, NY, USA, 2001. ACM. ISBN 1-58113-361-8. doi: http://doi.acm.org/10.1145/375551.375554. URL http://doi.acm.org/10.1145/375551.375554.

Y. Wang and M. Sakai. Decidability of termination for semi-constructor trss, left-linear shallow trss and related systems. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA)*, volume 4098 of *Lecture Notes in Computer Science*, pages 343–356. Springer, 2006.

C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. Spass version 3.5. In *22nd International Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009.

P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Proceedings of the 10th International Conference on Computer Aided Verification*, CAV '98, pages 88–97, London, UK, 1998. Springer-Verlag. ISBN 3-540-64608-6. URL http://portal.acm.org/citation.cfm?id=647767.733630.

K. Wong and C. Löding. Unranked tree automata with sibling equalities and disequalities. In *Proceedings of 34th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 4596 of *Lecture Notes in Computer Science*, pages 875–887. Springer, 2007. ISBN 978-3-540-73419-2.

N. B. Youssef, A. Bouhoula, and F. Jacquemard. Automatic verification of conformance of firewall configurations to security policies. In *Proceedings of the 14th IEEE Symposium on Computers and Communications (ISCC)*, pages 526–531. IEEE, 2009.

H. Zhang, D. Kapur, and M. S. Krishnamoorthy. A mechanizable induction principle for equational specifications. In *In Proc. 9th Int. Conf. on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 1988.

Index