# A Logic You Can Count On

Silvano Dal Zilio, Denis Lugiez and Charles Meyssonnier

LIF, Laboratoire d'Informatique Fondamentale de Marseille
CNRS and Université de Provence

## Abstract

We prove the decidability of the quantifier-free, static fragment of ambient logic, with composition adjunct and iteration, which corresponds to a kind of regular expression language for semistructured data. The essence of this result is a surprising connection between formulas of the ambient logic and counting constraints on (nested) vectors of integers.

Our proof method is based on a new class of tree automata for unranked, unordered trees, which may result in practical algorithms for deciding the satisfiability of a formula. A benefit of our approach is to naturally lead to an extension of the logic with recursive definitions, which is also decidable. Finally, we identify a simple syntactic restriction on formulas that improves the effectiveness of our algorithms on large examples.

**Categories and Subject Descriptors:** E.1 [**Data Structures**]: Trees; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*modal logic*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*classes defined by automata*; H.2.1 [**Database Management**]: Logical Design

**General Terms:** Algorithms, languages, theory, verification

**Keywords:** Ambient, substructural logic, semi-structured data, tree automata, Presburger arithmetic

## 1 Introduction

We prove the decidability of the static fragment of ambient logic [9], with composition adjunct and iteration, which corresponds to a kind of regular expression language for tree-like data structures.

The ambient logic is a modal logic proposed to describe the structural and behavioral properties of mobile ambients [8]. In this paper, we only consider the spatial fragment of the logic and work with finite, static processes. This static fragment, also called the *tree logic* (TL) in [7], is essentially a logic on finite edge-labeled trees. The study of TL is motivated by a connection with type systems and query languages for semistructured data [1] exploited by Cardelli and Ghelli in their language TQL. In their approach, a formula of TL may be considered as a simple yes/no query against a (tree representing a) database [5], where the answer is yes if the tree satisfies the formula. With some extensions, a formula may also be used to extract the subparts of a tree that match a description.

In this setting, we are interested by two problems: *model-checking*, to test whether a given information tree satisfies a formula; and *satisfiability*, to test if there exists a tree that satisfies a formula. Given the parallel between TL and query languages, model-checking appears similar to computing the result of a query, while satisfiability is useful for query optimizations or to check query inclusion (this problem is also related to subtyping in the implementation of TQL).

The models of the tree logic are terms of the form $a_1[d_1] \mid \cdots \mid a_p[d_p]$, called *information trees*, obtained by the parallel composition of a sequence of *elements*. Elements have a name (label), $a$, and a value (they lead to a subtree), $d$. Intuitively, information trees are nested multisets of labels and may be compared to XML documents, where elements are of the form `<a>`$d$`<\a>`, except that the order of elements in a tree is not relevant. The tree logic is equally uncluttered and includes primitives for tree composition, $A \mid B$, for element traversing, $a[A]$, and the implication induced by composition, $A \triangleright B$, with a simple and intuitive meaning: *composition*, $A \mid B$, is satisfied by trees $d_1 \mid d_2$ where $d_1$ satisfies $A$ and $d_2$ satisfies $B$; *location*, $a[A]$, is satisfied by trees with a single element $a[d]$ where $d$ satisfies $A$; *composition adjunct*, $A \triangleright B$, is satisfied by trees that, when composed with any tree that satisfies $A$, result in trees that satisfy $B$.

The decidability of the model-checking and satisfiability problems is not trivial. Indeed, the meaning of $A \triangleright B$ is defined through a possibly infinite quantification over the set of trees satisfying $A$. Another difficulty arises from indefinite repetition (Kleene star) $A^*$, which is defined as a form of fixed point on the horizontal structure of a tree. We prove the decidability of the model-checking and satisfiability problems for TL, as well as the decidability for the logic enriched with a limited form of fixed point on the vertical structure of a tree, akin to path expressions, and show that these two kinds of recursion are indeed orthogonal.

## Motivation and Related Work

Our motivations for this work stand at the intersection of two long-term research projects in which the authors are involved: a project concerned with the study of logical systems for mobile distributed systems and a project related to languages for manipulating semistructured data.

In the context of the first research project [27], our goal is to improve our knowledge on the complexity of the ambient logic. The choice of ambient logic is pertinent because it gives a general language for expressing behaviors of spatially distributed systems and because its lack of sensitivity to the details of the underlying model makes it easily transposable to other settings (such as the $\pi$-calculus [6] or almost every calculus with a system of nested locations). For the same reasons, it is also a perfect test bed for extensions, such as quantification on fresh names [10, 20].

We only consider the static fragment of ambient logic on finite processes. Previous works have shown that the model checking problem is PSPACE for the logic without adjunct [11] and that it is undecidable for the logic with name quantification and composition adjunct [12]. In [16], the authors show decidability of the satisfiability problem for the logic without adjunct and name quantification using tree automata and a logic on finite multisets. The result is extended to the logic with adjunct in [3]. The method used is adapted from a technique for proving decidability of validity in a spatial logic for reasoning about heaps [4] and is based on finite test sets for $\triangleright$. Since the size of a test set is not elementary in the size of the formula (it is not bounded by any tower of exponentials) it is not obvious that this approach may lead to a practical algorithm.

In this paper, we prove the decidability of the tree logic with adjunct, iteration and a restricted form of recursion along the paths of a tree. Our proof method is based on a surprising relation between TL and arithmetical constraints on vectors of integers (expressed as formulas of Presburger arithmetic). To obtain our decidability results, we show the equivalence between TL and a new logic on nested multisets of labels, the *sheaves logic*, that directly includes Presburger arithmetic formulas. In our approach, the sheaves logic appears more amenable to automatic processing and plays the role of a target (assembly) language in which we compile formulas of the tree logic.

The second research project is related to languages for manipulating semistructured data. As remarked by Cardelli and Ghelli [7], the tree logic is analogous to a regular expression language for tree-like data structures and is therefore a perfect basis for typing languages manipulating semistructured data. We can draw a parallel with the use of regular tree expressions in the language XDuce [21], where a logic similar to TL, albeit on an ordered model, is used to type extended pattern matching operations over XML documents.

The algorithmic methods used in the implementation of XDuce are based on regular tree automata [14]. Unfortunately, regular tree automata are not well-suited for unranked or unordered trees. For example, regular tree languages are generally not closed under associativity or associativity-commutativity (AC) of function symbols. In this paper, we use a simple extension of regular tree automata that works on information trees. This class of automata, called *sheaves automata*, is expressive enough to accept the set of trees matched by an ambient logic formula.

The definition of sheaves automata may be generalized to an algebra with an arbitrary number of free function symbols and with any number of associative and AC operators [24]. For example, an extended version of sheaves automata has been used by the authors to prove decidability results on a fragment of XML schema [17]. Therefore, the logics and the results given in this paper may be extended to a tree model with both sequential and parallel composition operators.

## Outline and Contributions

The paper is organized as follows. In Section 2 and 3 we review background material on information trees, on the Tree Logic (TL) and on Presburger arithmetic. In Section 4 we define a new modal logic for information trees, the Sheaves Logic, which is based on an alternative representation of multisets as the product of a sequence of multiplicities with a sequence of elements. As it is often the case, the shift in the data-structure makes it possible to use more elaborate algorithmic methods. At this point, we can already show that the complexity of the logic results equally from the use of composition adjunct as from the combination of composition with negation. More surprisingly, we identify iteration as the “most expensive” primitive.

In Section 5, we show how to interpret every connector of TL in the sheaves logic (SL). As a result, we obtain a compositional encoding of TL in SL. The idea is to prove the decidability of SL instead of directly studying TL. To this end, we define (Section 6) a new class of tree automata specifically designed for manipulating sheaves. This class of automata works directly on information trees; it is closed by the classical boolean operations and by tree composition; and it has a decidable test for emptiness.

Before concluding, we exploit the inherent recursiveness of automata and augment the (sheaves and tree) logics with a limited form of recursive definitions. This extension is expressive enough to include path expressions. A limitation of our method is that trees must be processed bottom-up, which may be very inefficient in the case of large trees. To avoid this problem, we identify a simple syntactical restriction, borrowed from a constraint found in XML Schema [30], that allows for the use of a top-down version of sheaves automata.

The connection between Presburger arithmetic and multiset logics was already used in previous work by the authors [16, 17]. In this paper, we clear up the relevance of this approach in the case of TL and extend our results to a fragment of the ambient logic with composition adjunct, $\triangleright$, and Kleene star. It is also the first time that we consider an extension of the logic with mutual recursive definitions. Another contribution of this work is to propose an approach more directed towards practical algorithms, for instance through the definition of *bases* (see Section 4 and Appendix A), with a study of the algorithmic complexity of our methods and the definition of possible simplifying restrictions.

Omitted proofs may be found in a long version of this paper [18].

## 2 Information Trees and the Tree Logic

Our model for semi-structured data is borrowed from [7]. Information trees [7] provide a compact syntax for defining nested multisets of labels borrowed from the ambient calculus [8]. They correspond to the static fragment of the ambient calculus, without primitives for mobility, communication and name scoping – but the same fragment may be found in almost every mobile process calculus with systems of nested locations. The resulting model is very close to the XML document model, with the difference that the order of the fields (subtrees) in an information tree is irrelevant. More formally, tree composition is an associative and commutative operator.

The following table summarizes the syntax of information trees. Given a set $\Lambda$ of element labels, we define the set $\mathcal{E}$ of elements and the set $\mathcal{IT}$ of information trees.

**Elements and Information Trees**

| | |
|---|---|
| $e ::=$ | element |
| $a[d]$ | element labeled $a$ (with $a \in \Lambda$), containing $d$ |
| $d ::=$ | information tree |
| $\mathbf{0}$ | empty information tree |
| $e$ | element |
| $d \mid d'$ | composition |

Trees with an equivalent structure are identified. This is expressed by means of a *structural congruence*, the smallest relation on $\mathcal{IT} \times \mathcal{IT}$ that is a congruence and such that $d \mid \mathbf{0} \equiv d$, and $d \mid d' \equiv d' \mid d$ and $d \mid (d' \mid d'') \equiv (d \mid d') \mid d''$. This relation coincides with structural congruence for the finite, static fragment of the ambient calculus. In the remainder of this paper, we work with terms modulo structural equivalence. Hence, we view information trees as nested multisets of elements. The $\mathbf{0}$ process is often omitted in the context $a[\mathbf{0}]$, yielding $a[\,]$.

*Example 1.* The following information tree may be interpreted as a valid entry for the bibliographical reference [8]:

$$\begin{array}{l} \mathit{article}[\,\mathit{title}[\texttt{Mobile Ambients}[\,]] \mid \mathit{author}[\texttt{Cardelli}[\,]] \\ \qquad \mid \mathit{author}[\texttt{Gordon}[\,]] \mid \mathit{year}[\texttt{1998}[\,]]\,]\ . \end{array}$$

To reason about the spatial and temporal properties of mobile ambients, Cardelli and Gordon have introduced the *modal logic of ambients* [9]. The static fragment of the ambient logic, which only refers to the spatial distribution of locations, appears particularly well-suited to describe the structure of information trees. In this paper, we study the static fragment of ambient logic, also called the Tree Logic, TL. A distinctive feature of the logic considered here, compared to [3], is that we enrich the syntax with an operator for indefinite repetition, $A^*$, which captures a simple class of recursive formulas. With this operator, TL appears as a kind of regular expression language, albeit on an unordered data model.

**Tree Logic Formulas**

| | |
|---|---|
| $A, B ::=$ | formula |
| $\top$ | true |
| $\neg A$ | negation |
| $A \vee B$ | disjunction |
| $\mathbf{0}$ | empty tree |
| $a[A]$ | location ($a \in \Lambda$) |
| $A \mid B$ | composition |
| $A \triangleright B$ | composition adjunct |
| $A^*$ | iteration |

The denotation of a formula $A$ is a set $[\![A]\!]$ of information trees. As usual, we say that a tree satisfies a formula, denoted $d \models A$, if and only if $d \in [\![A]\!]$.

To simplify the presentation, we extend the composition operator to sets of trees, $\mathcal{S} \mid \mathcal{S}' =_{\text{def}} \{(d \mid d') \mid (d, d') \in \mathcal{S} \times \mathcal{S}'\}$. With this definition, it is easy to show that the structure $(2^{\mathcal{IT}}, \cup, \emptyset, |, \{\mathbf{0}\})$ is a semi-ring, where $2^{\mathcal{IT}}$ is the powerset of $\mathcal{IT}$ (modulo $\equiv$). Sticking to the multiplicative interpretation of composition, we use the notation $\mathcal{S}^n$ for the composition $\mathcal{S} \mid \cdots \mid \mathcal{S}$ ($n$ times), $\mathcal{S}^0$ for the singleton $\{\mathbf{0}\}$, and $\mathcal{S}^*$ for the set $\bigcup_{n \geqslant 0} \mathcal{S}^n$. Finally, if $\mathbf{n} = (n_1, \ldots, n_p)$ is a sequence of integers, and $\mathbf{S} = (S_1, \ldots, S_p)$ is a sequence of sets of elements, we write $\mathbf{n} \centerdot \mathbf{S}$ for the set $S_1^{n_1} \mid \ldots \mid S_p^{n_p}$. The latter notation will sometimes be referred to as a *sheaved composition*, and is at the core of the new logic presented in this paper.

**Satisfaction**

| | | |
|---|---|---|
| $[\![\top]\!]$ | $=$ | $\mathcal{IT}$ |
| $[\![\neg A]\!]$ | $=$ | $\mathcal{IT} \setminus [\![A]\!]$ |
| $[\![A \vee B]\!]$ | $=$ | $[\![A]\!] \cup [\![B]\!]$ |
| $[\![\mathbf{0}]\!]$ | $=$ | $\{\mathbf{0}\}$ |
| $[\![a[A]]\!]$ | $=$ | $\{a[d] \mid d \in [\![A]\!]\}$ |
| $[\![A \mid B]\!]$ | $=$ | $[\![A]\!] \mid [\![B]\!]$ |
| $[\![A \triangleright B]\!]$ | $=$ | $\{d \mid \forall d' \in [\![A]\!] \,.\, (d \mid d') \in [\![B]\!]\}$ |
| $[\![A^*]\!]$ | $=$ | $[\![A]\!]^*$ |

The satisfaction rules for the propositional fragment are conventional. The formula $\mathbf{0}$ only matches (trees structurally equivalent to) $\mathbf{0}$, the location $a[A]$ matches $d$ if $d \equiv a[d']$ with $d' \models A$, and the composition $A_1 \mid A_2$ matches $d$ if $d \equiv d_1 \mid d_2$ with $d_i \models A_i$ for all $i \in \{1, 2\}$.

*Example 2.* The following formula of TL matches "valid" bibliographical entries, like the term given in Example 1:

$$\begin{array}{l} \mathit{article}[\,\mathit{title}[\top] \mid \mathit{author}[\top] \\ \qquad\qquad \mid \neg((\mathit{title}[\top] \vee (\mathit{year}[\top] \mid \mathit{year}[\top])) \mid \top)\,]\ . \end{array}$$

This formula specifies that a valid bibliographical entry must contain *exactly* one field labeled *title*, *at least* one field labeled *author*, and *at most* one field labeled *year*, possibly alongside some other (unspecified) fields. These constraints could be expressed more directly using sheaved composition, by saying that a valid entry must be in one of the sets:

$$(n_t, n_a, n_y, n_o) \centerdot ([\![\mathit{title}[\top]]\!], [\![\mathit{author}[\top]]\!], [\![\mathit{year}[\top]]\!], S_o)\ ,$$

where $n_t, n_a, n_y, n_o$ are integer variables such that $(n_t = 1)$ and $(n_a \geqslant 1)$ and $(n_y \leqslant 1)$ and $(n_o \geqslant 0)$, and $S_o$ is the set of elements of the form $x[d]$ with $x \notin \{\mathit{title}, \mathit{author}, \mathit{year}\}$.

One of the main contributions of this paper is to show that any quantifier-free TL-formula can be expressed in the same way, that is, as the product of integers vectors (definable in Presburger arithmetics) by sequences of element formulas.

In the next section, we recall the background on Presburger arithmetic and semilinear sets that we need in order to define the Sheaves Logic (SL) formally, and then to prove its equivalence with the quantifier-free fragment of TL.

## 3 Background

This section collects the necessary background on Presburger arithmetic and its connection with semilinear sets. Presburger arithmetic is the first-order theory of equality over the group $(\mathbb{N}, +)$ of natural numbers with addition. Presburger formulas (also called constraints) are described in the following table, where $M, N, \ldots$ range over integer variables and $m, n, \ldots$ range over integer constants.

**Presburger Constraints**

| | |
|---|---|
| $\mathit{Exp} ::=$ | Integer expression |
| $n$ | positive integer constant |
| $N$ | positive integer variable |

| | |
|---|---|
| $Exp_1 + Exp_2$ | addition |
| $\phi, \psi, \ldots ::=$ | Presburger arithmetic formulas |
| $(Exp_1 = Exp_2)$ | test for equality |
| $\neg\phi$ | negation |
| $\phi \vee \psi$ | disjunction |
| $\exists N.\phi$ | existential quantification |

Presburger constraints may be used to define a substantial class of (decidable) properties over positive integers, like for example "the value of $M$ is strictly greater than the value of $N$", using the formula $\exists X.(M = N + X + 1)$; or "$M$ is an odd number", $\exists X.(M = X + X + 1)$. In this paper, we use Presburger formulas to express arithmetical constraints over multiplicities of multisets of elements.

Throughout the text we use the vector notation, $\mathbf{n}$, for tuples of integers, and $|S|$ for the size (number of elements) of $S$. We denote $\phi(\mathbf{N})$ a Presburger formula whose free variables are all in $\mathbf{N} = (N_1, \ldots, N_p)$ and we use the notation $\models \phi(n_1, \ldots, n_p)$ when $\phi\{N_1 \leftarrow n_1\} \ldots \{N_p \leftarrow n_p\}$ is satisfied.

The denotation $[\![\phi(\mathbf{N})]\!]$ of a Presburger formula $\phi(\mathbf{N})$ is the set of integer vectors $\mathbf{n}$ such that $\models \phi(\mathbf{n})$. Presburger arithmetic is an interesting example in computational complexity theory because it is one of the few problem that provably need more than polynomial run time [19]: every algorithm which decides the truth of a Presburger constraint $\phi$, that is test whether $[\![\phi]\!] = \emptyset$, has a runtime of at least $2\hat{\ }(2\hat{\ }(cn))$ for some constant $c$, where $n$ is the length of $\phi$. (The expression $a \hat{\ } b$ stands for the exponentiation $a^b$.) There is also a known triply exponential upper-bound in the worst case [26], that is for an unbounded alternation of quantifiers: the complexity of checking the satisfiability of a formula $\phi$ is in time at most $2\hat{\ }(2\hat{\ }(2\hat{\ }(pn)))$. The problem is NP-complete for the existential fragment of Presburger arithmetic.

## 3.1 Semilinear Sets

Decidability of Presburger arithmetic may be proved using a connection with *semilinear sets* of natural numbers. A *linear set* of $\mathbb{N}^n$, $L(\mathbf{b}, P)$, is a set of vectors generated by linear combination of the periods $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_k\}$ (with $\mathbf{p}_i \in \mathbb{N}^n$ for all $i \in 1..p$), with the base $\mathbf{b} \in \mathbb{N}^n$:

$$L(\mathbf{b}, P) =_{\text{def}} \{\mathbf{b} + \sum_{i \in 1..k} \lambda_i \mathbf{p}_i \mid \lambda_1, \ldots, \lambda_k \in \mathbb{N}\} .$$

A *semilinear set* is a finite union of linear sets. Semilinear sets are closed under set operations and are exactly the models of Presburger arithmetic formulas, that is, the set of integer vectors satisfying a formula $\phi(N_1, \ldots, N_p)$ is a semilinear set of $\mathbb{N}^p$ and conversely. An important result is that semilinear sets are also closed under the following typical operators of regular word languages: $L + M =_{\text{def}} \{x + y \mid x \in L, y \in M\}$, $L^n =_{\text{def}} L + \ldots + L$ ($n$ times), and $L^* =_{\text{def}} \bigcup_{n \geq 0} L^n$.

PROPOSITION 3.1 (FROM [16]). *For any two semilinear sets $L, M$ of $\mathbb{N}^p$, the sets $L + M$, $L^k$ ($k \in \mathbb{N}$) and $L^*$ are also semilinear sets of $\mathbb{N}^p$.*

Presburger constraints and semilinear sets are *effectively* equivalent, that is, given a Presburger constraint $\phi$, it is possible to compute the bases and periods of a semilinear set representing $[\![\phi]\!]$, and conversely [15]. To build a semilinear set corresponding to a given Presburger constraint, it is enough to perform quantifier elimination on the constraint, a possibly expensive procedure.

## 3.2 Sum, Iteration and Derived Connectors

Using the relation between semilinear sets and Presburger formulas, we can lift sum and iteration to the level of the logic. We also make use of the usual derived connectives, *conjunction* $\wedge$, *implication* $\rightarrow$, and *universal quantification* $\forall$. The essence of our equivalence result (Section 5) is a connection between sum and tree composition. The sum of two formulas, $\phi(\mathbf{N}) + \psi(\mathbf{N})$, is a formula such that $[\![\phi + \psi]\!] = [\![\phi]\!] + [\![\psi]\!]$.

Assume $\mathbf{N}, \mathbf{N}_1$ and $\mathbf{N}_2$ are disjoint sequences of variables.

$$(\phi + \psi)(\mathbf{N}) =_{\text{def}} \exists \mathbf{N}_1, \mathbf{N}_2.(\mathbf{N} = \mathbf{N}_1 + \mathbf{N}_2) \wedge \phi(\mathbf{N}_1) \wedge \psi(\mathbf{N}_2)$$

As in the ambient logic, we may define the adjunct and the dual operators for sum. We use the same notation as in TL for the adjunct, $\triangleright$, such that $\phi + \psi \vdash \xi$ if and only if $\phi \vdash \psi \triangleright \xi$ and we denote $\|$ the DeMorgan dual of sum, defined by the relation $\phi \| \psi =_{\text{def}} \neg(\neg\phi + \neg\psi)$. (The entailment relation, $\phi \vdash \psi$, means that $[\![\phi]\!] \subseteq [\![\psi]\!]$.) The following definition makes clear the correspondence between $\triangleright$ and the linear implication connector of linear logic, which was already mentioned in [9].

Assume $\mathbf{M}, \mathbf{N}, \mathbf{N}_1, \mathbf{N}_2$ are disjoint sequences of variables.

$$(\phi \triangleright \psi)(\mathbf{N}) =_{\text{def}} \forall \mathbf{M}.\, \phi(\mathbf{M}) \rightarrow \psi(\mathbf{N} + \mathbf{M})$$

$$(\phi \,\|\, \psi)(\mathbf{N}) =_{\text{def}} \forall \mathbf{N}_1, \mathbf{N}_2.\, (\mathbf{N} = \mathbf{N}_1 + \mathbf{N}_2) \rightarrow \phi(\mathbf{N}_1) \vee \psi(\mathbf{N}_2)$$

The following property states the soundness of these encodings and relates $\triangleright$ with a subtraction operation over the powerset of $\mathbb{N}^p$.

PROPOSITION 3.2. *Assume $\phi$ and $\psi$ are two Presburger formulas with the same free variables, then for every set of integer vectors, $\mathcal{S}$, we have:*

$$\begin{array}{lcl} \mathcal{S} \subseteq [\![\phi + \psi]\!] & \Leftrightarrow & \mathcal{S} \subseteq [\![\phi]\!] + [\![\psi]\!], \\ \mathcal{S} \subseteq [\![\phi \triangleright \psi]\!] & \Leftrightarrow & \mathcal{S} + [\![\phi]\!] \subseteq [\![\psi]\!] . \end{array}$$

It is also possible to derive a formula for iteration, $\phi^*$, such that $[\![\phi^*]\!] = [\![\phi]\!]^*$, that is, $\models \phi^*(\mathbf{n})$ if and only if $\mathbf{n}$ is the sum of a finite number of vectors satisfying $\phi$ (we take the empty sum to stand for the null vector, $\mathbf{0} = (0, \ldots, 0)$). Unlike the previous cases, the definition of $\phi^*$ is quite complex. One may find a possible construction in [16, Section 3], which requires to compute the basis and periods of the linear sets associated with $\phi$. The size of the formula $\phi^*$ may be exponentially bigger than the size of $\phi$.

Assume $\phi$ is a formula such that $[\![\phi]\!] = \bigcup_{i \in 1..l} L(\mathbf{b}_i, P_i)$, with $P_i = \{\mathbf{p}_{i,1}, \ldots, \mathbf{p}_{i,l_i}\}$ :

$$\begin{array}{lll} \phi^*(\mathbf{N}) & =_{\text{def}} & \exists \mu_i, \lambda_{i,j}.(\mathbf{N} = \sum_{i \leqslant l} (\mu_i \mathbf{b}_i + \sum_{j \leqslant l_i} \lambda_{i,j} \mathbf{p}_{i,j})) \\ & & \wedge \bigwedge_{i \leqslant l} ((\bigvee_{j \leqslant l_i} \lambda_{i,j} \neq 0) \rightarrow \mu_i \neq 0) \end{array}$$

In the ambient logic, computing the denotation of $A \triangleright B$ requires a universal quantification over $[\![A]\!]$ and it is therefore a costly operation. This complexity issue does not appear as clearly in the definition of subtraction. Given that an uncontrolled alternation of $+$ and $\|$ produces Presburger formulas with an unbounded alternation of quantifiers, the use of $\triangleright$ is not more problematic than the combination of composition (+) and negation. More surprisingly, when considering operators derived from TL, it appears that the complexity is dominated by iteration.

## 4 The Sheaves Logic

The Sheaves Logic (SL) is a new modal logic for information trees that directly encompasses Presburger constraints. The logic, summarized below, is built upon a very limited set of modalities and does not even directly embed propositional logic. Nonetheless, we show in the next section how we can derive all the modalities of TL.

**SL-Formulas**

| | |
|---|---|
| $\alpha ::=$ | Label expression |
| $a_1, \dots, a_n$ | finite subset of $\Lambda$ |
| $\alpha^\perp$ | complement of $\alpha$ |
| $E ::=$ | Element formula |
| $\alpha[A]$ | element with label in $\alpha$ |
| $A ::=$ | Counting formula |
| $\top$ | true |
| $\exists \mathbf{N}. \phi(\mathbf{N}) . \mathbf{E}$ | sheaves composition (with $\lvert\mathbf{N}\rvert = \lvert\mathbf{E}\rvert$) |

We use uppercase letters $A, B, \dots$ to denote formulas of SL, but this should not cause any ambiguity with TL. We refer to the sequence, $\mathbf{E}$, of element formulas appearing in a sheaves composition, $A$, as the *support vector* of $A$. Informally, an element formula $\alpha[A]$ in a support vector matches groups of elements of the form $a[d]$, with $a \in \alpha$. A composition, $\exists \mathbf{N}. \phi . \mathbf{E}$, is a quantification over the number of elements in each of these groups, constrained by the Presburger formula $\phi$. The last formula, $\top$, does not constrain its model in any way.

The meaning of SL-formulas is defined by means of a satisfaction relation. To simplify the presentation, we extend sheaves composition to sequences of sets, where $(n_1, \dots, n_p) . (S_1, \dots, S_p)$ stands for the set $S_1^{n_1} \mid \cdots \mid S_p^{n_p}$.

**Satisfaction**

| | | |
|---|---|---|
| $[\![a_1, \dots, a_n]\!]$ | $=$ | $\{a_1, \dots, a_n\}$ |
| $[\![\alpha^\perp]\!]$ | $=$ | $\Lambda \setminus [\![\alpha]\!]$ |
| $[\![\alpha[A]]\!]$ | $=$ | $\{a[d] \mid a \in [\![\alpha]\!] \wedge d \in [\![A]\!]\}$ |
| $[\![(E_1, \dots, E_p)]\!]$ | $=$ | $([\![E_1]\!], \dots, [\![E_p]\!])$ |
| $[\![\top]\!]$ | $=$ | $\mathcal{IT}$ |
| $[\![\exists \mathbf{N}. \phi(\mathbf{N}) . \mathbf{E}]\!]$ | $=$ | $\bigcup_{\mathbf{n} \in [\![\phi]\!]} \mathbf{n} . [\![E]\!]$ |

Label expressions represent finite and co-finite sets of elements. An interesting example of label expression is $\emptyset^\perp$, that matches every possible label. We use the notation AnyE for the element formula $\emptyset^\perp[\top]$, matching every element in $\mathcal{E}$.

The semantic definition of sheaves composition is probably easier to understand in terms of the associated satisfaction relation. Assume $\mathbf{E}$ is the support $(E_1, \dots, E_p)$ and $\mathbf{n} = (n_1, \dots, n_p)$. An information tree is in the set $\mathbf{n} . [\![E]\!]$ if and only if it may be decomposed into the product of $n_1$ elements satisfying $E_1$, ..., and $n_p$ elements satisfying $E_p$.

$$d \in \mathbf{n} . [\![\mathbf{E}]\!] \Leftrightarrow \begin{cases} d \equiv \prod_{i \in 1..p} (e_1^i \mid \cdots \mid e_{n_i}^i) \\ e_j^i \models E_i \text{ for all } i \in 1..p, j \in 1..n_i \end{cases}$$

Then $d$ satisfies $\exists \mathbf{N}. \phi . \mathbf{E}$ if and only if there exists a sequence of multiplicities, $\mathbf{n}$, such that $\models \phi(\mathbf{n})$ and $d \in \mathbf{n} . [\![E]\!]$.

The syntax of SL does not restrict the set of support vectors that may be used in a formula. Nonetheless, some supports have better properties than others. For example, a support vector may be *generating*, that is, $\bigcup_{\mathbf{n} \in \mathbb{N}^p} \mathbf{n} . [\![E]\!] = \mathcal{IT}$, or its element formulas may have disjoint interpretations. In the latter case, we say that the element formulas are *linearly independent*. This property is interesting since, in this case, it can be proved that the decomposition of a tree is always unique, that is, $\mathbf{n} . [\![E]\!] \cap \mathbf{m} . [\![E]\!] \neq \emptyset$ if and only if $\mathbf{n} = \mathbf{m}$. Drawing a parallel with linear algebra, we define a notion of "good" support vectors, that we call *bases*, which are maximal sequences of linearly independent element formulas. We fulfill the canonical property of linear algebra: every information tree admits a unique decomposition following a given basis.

DEFINITION 4.1 (BASIS). *A vector $(E_1, \dots, E_p)$ is a basis if and only if $i \neq j$ implies $[\![E_i]\!] \cap [\![E_j]\!] = \emptyset$ for all $i, j \in 1..p$ and $\bigcup_{i \in 1..p} [\![E_i]\!] = \mathcal{E}$. A basis $\mathbf{E}$ is proper if and only if every support vector appearing in a subformula of $\mathbf{E}$ is also a basis.*

The simplest example of (proper) basis is the singleton sequence AnyE. Another simple example is the sequence $(a_1[\top], \dots, a_p[\top], \Sigma^\perp[\top])$, where $\Sigma = \{a_1, \dots, a_p\}$ is a finite subset of $\Lambda$.

## 5 Encoding TL in SL

In this section, we define derived formulas for every operator of TL. As a result, we obtain a compositional encoding from TL to SL that preserves the interpretation of formulas.

### 5.1 Derived Operators

Let $A$ be the composition $\exists \mathbf{N}. \phi . \mathbf{E}$. If the counting constraint is only satisfied by the null vector then $A$ only matches $\mathbf{0}$. Likewise, if $\phi$ is a tautology and $\mathbf{E}$ is generating then $A$ matches every tree in $\mathcal{IT}$. Finally, we can easily encode the location formula, $\alpha[A]$, using as support vector the (size 1) sequence $\alpha[A]$.

| | | |
|---|---|---|
| $\mathbf{0}$ | $=_{\text{def}}$ | $\exists N. (N = 0) . \text{AnyE}$ |
| True | $=_{\text{def}}$ | $\exists N. (N \geqslant 0) . \text{AnyE}$ |
| $\alpha[A]$ | $=_{\text{def}}$ | $\exists N. (N = 1) . \alpha[A]$ |

The next proposition states that our encoding is faithful to the meaning of these operators in TL.

PROPOSITION 5.1. *The following three equations hold:*

$$\begin{aligned} [\![\mathbf{0}]\!] &= \{\mathbf{0}\}, \\ [\![\alpha[A]]\!] &= \{a[d] \mid a \in \alpha \wedge d \in [\![A]\!]\}, \\ [\![\text{True}]\!] &= [\![\top]\!] \;=\; \mathcal{IT} \;. \end{aligned}$$

We can as easily transfer the "positive" composition operators of TL (disjunction, parallel composition and iteration) to SL if we assume that the formulas use the same support.

Assume $A = \exists \mathbf{N}. \phi_A . \mathbf{E}$ and $B = \exists \mathbf{N}. \phi_B . \mathbf{E}$.

| | | |
|---|---|---|
| $A \vee B$ | $=_{\text{def}}$ | $\exists N. (\phi_A \vee \phi_B) . \mathbf{E}$ |
| $A \mid B$ | $=_{\text{def}}$ | $\exists N. (\phi_A + \phi_B) . \mathbf{E}$ |
| $A^*$ | $=_{\text{def}}$ | $\exists N. (\phi_A^*) . \mathbf{E}$ |

The soundness of these encodings is based on the algebraic properties of the semi-ring $(2^{\mathcal{IT}}, \cup, \emptyset, |, \{\mathbf{0}\})$, like distributivity of parallel composition over set union, for the encoding of disjunction, and the exponentiation rule, $S^{n_1} \mid S^{n_2} = S^{n_1+n_2}$, for the encoding of composition.

PROPOSITION 5.2. *Assume $A = \exists \mathbf{N}. \phi_A . \mathbf{E}$ and $B = \exists \mathbf{N}. \phi_B . \mathbf{E}$ then the following equations hold:*

$$\begin{aligned} [\![A \vee B]\!] &= [\![A]\!] \cup [\![B]\!], \\ [\![A \mid B]\!] &= [\![A]\!] \mid [\![B]\!], \\ [\![A^*]\!] &= [\![A]\!]^* \ . \end{aligned}$$

Encoding "negative" operators is not as simple. In particular, the complement of a composition, $A = \exists \mathbf{N}. \phi . \mathbf{E}$, is not necessarily the formula $A^\perp = \exists \mathbf{N}. (\neg\phi) . \mathbf{E}$, obtained by complementing the counting constraint. Possible sources of problems are non-generating support vector. A simple example illustrates this case. Let $A$ be the composition $\exists N. (N = 0) . a[\top]$, with model $\{\mathbf{0}\}$. Then $A^\perp = \exists N. (N \neq 0) . a[\top]$ matches trees with only elements labeled $a$ at top-level, but does not match the tree $a[\,] \mid b[\,]$.

In the case of sheaves formulas built from a common basis, we can interpret negation and composition adjunct in a direct manner.

Assume $A = \exists \mathbf{N}. \phi_A . \mathbf{E}$ and $B = \exists \mathbf{N}. \phi_B . \mathbf{E}$ where $\mathbf{E}$ is a basis.

$$\begin{aligned} \neg A &=_{\text{def}} \exists N. (\neg\phi_A) . \mathbf{E} \\ A \wedge B &=_{\text{def}} \exists N. (\phi_A \wedge \phi_B) . \mathbf{E} \\ A \triangleright B &=_{\text{def}} \exists N. (\phi_A \triangleright \phi_B) . \mathbf{E} \end{aligned}$$

The soundness of these encodings directly relies on the canonical property of bases, that is, if $\mathbf{E}$ is a basis then for every tree $d$ there is a unique vector of multiplicities, $\mathbf{n}$, such that $d \in \mathbf{n} . [\![E]\!]$. In the case of negation, for example, we obtain $d \in [\![\exists \mathbf{N}. \phi . \mathbf{E}]\!]$ if and only if $\models \phi(\mathbf{n})$, as needed.

PROPOSITION 5.3. *Assume $A = \exists \mathbf{N}. \phi_A . \mathbf{E}$ and $B = \exists \mathbf{N}. \phi_B . \mathbf{E}$, with $\mathbf{E}$ a basis, then:*

$$\begin{aligned} [\![\neg A]\!] &= \mathcal{IT} \setminus [\![A]\!], \\ [\![A \wedge B]\!] &= [\![A]\!] \cap [\![B]\!], \\ [\![A \triangleright B]\!] &= \{d \mid \forall d' \in [\![A]\!], (d \mid d') \in [\![B]\!]\} \ . \end{aligned}$$

Given the encoding of negation, it is possible to define a new encoding of location built over a basis.

Assume $A = \exists \mathbf{N}. \phi_A . \mathbf{E}$ where $\mathbf{E}$ is a basis.

$$\begin{aligned} \alpha[A] =_{\text{def}} \ & \exists N_1, N_2, N_3 . (N_1 = 1) \wedge (N_2 = N_3 = 0) \\ & . (\alpha[A],\ \alpha[\neg A],\ \alpha^\perp[\top]) \end{aligned}$$

Next, we prove our main result that for every formula of TL, there exists an equivalent formula in SL. To overcome a small technical difficulty with the encodings of the derived operators, namely that we should work with formulas defined on a common basis, we show that we can always operate on formulas defined on a common basis.

PROPOSITION 5.4 (SUPPORT REFINEMENT). *Given a sequence of formulas $(A_1, \ldots, A_n)$, where $A_i$ is defined over a proper basis for all $i \in 1..n$, we can build a sequence of formulas $(B_1, \ldots, B_n)$, defined on a common proper basis, such that $[\![A_i]\!] = [\![B_i]\!]$ for all $i \in 1..n$.*

PROOF. See Proposition A.3 in Appendix A. □

## 5.2 Equivalence Result

The main theorem of this section is obtained by assembling our different results.

THEOREM 5.1. *For any TL-formula A, there is a SL-formula B defined over a proper basis such that $[\![A]\!] = [\![B]\!]$.*

PROOF. By induction on the syntax of $A$. We use Propositions 5.1, 5.2 and 5.3 to translate every operator and, at each step, we use Proposition 5.4 to ensure that we work with formulas defined on a common proper basis. □

It is also possible to prove an inclusion in the other direction. (For the sake of brevity, we do not give the details of the reverse encoding here, but a complete proof may be found in [18].)

THEOREM 5.2. *For any SL-formula A, there is a TL-formula B such that $[\![A]\!] = [\![B]\!]$.*

Iteration is essential in the proof that TL contains SL: without iteration, SL is strictly more expressive than TL. Conversely, we have omitted existential quantification over names, $\exists x.A$, in the syntax of TL. Our approach does not easily extend to the logic with quantifier. (Since the ambient logic with universal quantifier and composition adjunct is undecidable [12], there is not much hope!) Nonetheless, we can encode a weaker form of quantification using label expressions. For instance, we may encode the formula $\exists x.x[A]$, with $x$ not occurring in $A$, using the element formula $\emptyset^\perp[A]$. Label expressions may also encode a limited form of fresh quantification [10, 20], $\mathsf{N}x.A$, with the intuitive meaning that $A$ is true for almost every name $x$, except a finite number. The idea is to replace every occurrences of $x$ by $\alpha^\perp$, where $\alpha$ is a set of labels containing the free names of $A$.

## 5.3 Examples

To illustrate our results, we use our approach to prove the validity of simple statements in TL.

Our first equation below states that a single element not labeled with $a$ is a "single-threaded" tree (a tree with exactly one branch at the root: $\neg\mathbf{0} \wedge \neg(\neg\mathbf{0} \mid \neg\mathbf{0})$) that is not an element of the form $a[d]$ (the formula $\neg a[\top]$).

$$a^\perp[\top] = \neg\mathbf{0} \wedge \neg(\neg\mathbf{0} \mid \neg\mathbf{0}) \wedge \neg(a[\top]) \tag{1}$$

We consider the basis $\mathbf{E} = (a[\top], a^\perp[\top])$ and only reason on the counting constraints. We use the variable $M$ for the number of elements matching $a[\top]$ and $N$ for the number of elements matching $a^\perp[\top]$. We have that $\neg(a[\top])$ corresponds to $\neg((M = 1) \wedge (N = 0))$. Likewise, $\neg\mathbf{0}$ corresponds to the formula $\neg((M = 0) \wedge (N = 0))$ (or equivalently $M + N \neq 0$) and $\neg\mathbf{0} \mid \neg\mathbf{0}$ corresponds to:

$$\begin{aligned} \exists M_1, N_1, M_2, N_2 . (M = M_1 + M_2) \wedge (N = N_1 + N_2) \wedge \\ (M_1 + N_1 \neq 0) \wedge (M_2 + N_2 \neq 0) \end{aligned}$$

which is equivalent to $M + N \geqslant 2$. By combining these three Presburger formulas, we obtain that the right-hand side of (1) corresponds to $(M + N = 1) \wedge \neg((M = 1) \wedge (N = 0))$, which is equivalent to $(M = 0) \wedge (N = 1)$, as needed.

The second equation states that a composition of elements named $a$ may not contain (at top-level) an element not labeled with $a$.

$$a[\top]^* = \neg(\top \mid a^\perp[\top]) \qquad (2)$$

We use the same basis than in the previous example and only concentrate on the counting constraints. The left-hand side of (2) translates to $((M=1)\wedge(N=0))^*$, that is, to $(M \geqslant 0)\wedge(N=0)$. For the right-hand side, $\top \mid a^\perp[\top]$ corresponds to:

$$\exists (M_i, N_i)_{i\in 1,2}.(M = M_1 + M_2)\wedge(N = N_1 + N_2)\wedge(N_2 \geqslant 1)$$

which is equivalent to $(M \geqslant 0)\wedge(N \geqslant 1)$, as needed.

These examples illustrate how we can simply reduce the reasoning on TL to pure arithmetical reasoning. Presburger arithmetic is amenable to automatic theorem proving: there exist several dedicated provers [2, 28] and many available "generic" theorem provers include a decision procedure for (at least a fragment of) Presburger arithmetic. Therefore, a possible application of our encoding is to directly assert, or infer, valid statements in TL.

In order to deal with more general problems, we need a flexible framework for reasoning on the models of SL-formulas. Using the classical connection between logic and automata theory, we propose in the next section a class of tree automata specifically targeted at the manipulation of sheaves formulas.

## 6 Sheaves Automata

Information trees are essentially trees modulo an associative-commutative (AC) theory, it is therefore natural to use tree automata to reason on them. Nonetheless, regular tree automata [14] are not satisfactory in the presence of AC operators, such as composition $|$, and we need to introduce an extended class of automata tailored to our need.

A (bottom-up) sheaves automaton $\mathcal{A}$ is a triple $\langle Q, Q_{\text{fin}}, R\rangle$ where $Q = \{q_1, \ldots, q_p\}$ is a finite set of states, $Q_{\text{fin}}$ is a set of final states included in $Q$, and $R$ is a set of transition rules. Transition rules are two kinds:

$$\begin{array}{ll} (1) & \alpha[q'] \to q \\ (2) & \phi(\#q_1, \ldots, \#q_p) \to q \end{array}$$

Type (1) rules correspond to transition rules in regular tree automata (we only have unary function symbols, $\alpha[.]$). A minor difference is that, in order to work with infinite sets of labels, we use label expressions instead of simple labels.

Type (2) rules allow to compute on nodes with an unbounded arity, arising from the composition of two or more information trees. In type (2) rules, $\phi$ is a Presburger formula with free variables $\#q_1, \ldots, \#q_p$ (one for each state in $Q$). Intuitively, $\#q_i$ is a variable that will be substituted by the number of occurrences of the state $q_i$ in a transition of the automata. A type (2) rule may fire if we have a term of the form $e_1 \mid \ldots \mid e_n$ such that $e_i$ leads to a state $q_{j_i} \in Q$ for all $i \in 1..n$, and $\models \phi(m_1, \ldots, m_n)$, where $m_i$ is the multiplicity of $q_i$ in the multiset $q_{j_1} \mid \cdots \mid q_{j_n}$. A particular example of transition is obtained if $\models \phi(0, \ldots, 0)$, in which case the rule $\phi \to q$ may fire for the null tree, **0**.

*Example 3.* Let $\mathcal{A}$ be the automaton with states $Q = \{q_a, q_b, q_s\}$, set of final states $Q_{\text{fin}} = \{q_s\}$ and the following transition rules:

$$a[q_s] \to q_a \qquad b[q_s] \to q_b \qquad (\#q_a = \#q_b)\wedge(\#q_s \geqslant 0) \to q_s$$

We show in Example 4, after defining the transition relation, that $\mathcal{A}$ accepts exactly the set of trees with as many $a$'s as $b$'s at each node, like for example $b[] \mid a[b[] \mid a[]]$.

### 6.1 Transition Relation

The *transition relation* of an automaton $\mathcal{A}$ is the transitive closure of the relation defined by the two following rules. We use the notation $\#_Q(q_{j_1} \mid \ldots \mid q_{j_n})$ for the multiplicities of the states of $Q$ in the multiset $q_{j_1} \mid \ldots \mid q_{j_n}$.

**Transition Relation: $\to$**

(type 1)

$$\frac{d \to q' \quad \alpha[q'] \to q \in R \quad a \in \alpha}{a[d] \to q}$$

(type 2)

$$\frac{\begin{array}{c} e_1 \to q_{j_1} \quad \ldots \quad e_n \to q_{j_n} \quad \phi \to q \in R \\ (n \neq 1) \qquad \#_Q(q_{j_1} \mid \ldots \mid q_{j_n}) \in [\![\phi]\!] \end{array}}{e_1 \mid \ldots \mid e_n \to q}$$

To avoid ambiguities, a side-condition in the rule for constrained transitions ensure that it cannot be applied to sequences, $a[d]$, with a single element. It could be possible to have only one kind of transition rule, but it would needlessly complicate our definitions and proofs without adding expressivity.

*Example 4.* Let $\mathcal{A}$ be the automaton defined in Example 3. Since the constraint in the type (2) rule of $\mathcal{A}$ is satisfied by $(0,0,0)$, we have that $\mathbf{0} \to q_s$. Let $d$ be the tree $a[] \mid b[a[] \mid b[]]$, a possible accepting run of the automaton is given below:

$$\begin{array}{llllll} d & \to & a[\mathbf{0}] \mid b[a[q_s] \mid b[\mathbf{0}]] & \to & a[q_s] \mid b[a[q_s] \mid b[\mathbf{0}]] & \\ & \to & a[q_s] \mid b[a[q_s] \mid b[q_s]] & \to & q_a \mid b[a[q_s] \mid b[q_s]] & \\ & \to & q_a \mid b[a[q_s] \mid q_b] & \to & q_a \mid b[q_a \mid q_b] & \\ & \overset{\dagger}{\to} & q_a \mid b[q_s] & \to & q_a \mid q_b & \overset{\dagger}{\to} q_s \end{array}$$

In transitions 7 and 9 (marked with a †-symbol), we use the only constrained rule of $\mathcal{A}$. In each case, the multiset used in the constraints is $q_a \mid q_b$, which contains as many $q_a$'s than $q_b$'s (that is, $\#_Q(q_a \mid q_b) = (1,1,0)$).

We say that an automaton $\mathcal{A}$ is *deterministic* if and only if for every pair of distinct type (1) rules, $\alpha[q] \to q_1$ and $\beta[q] \to q_2$, we have $[\![\alpha]\!] \cap [\![\beta]\!] = \emptyset$ and for every pair of distinct type (2) rules, $\phi \to q_1$ and $\psi \to q_2$, we have $[\![\phi]\!] \cap [\![\psi]\!] = \emptyset$. A property of deterministic automata is that for each tree $d$ there is at most one state $q \in Q$ such that $d \to q$. As usual, we say that a tree $d$ is *accepted* by an automaton $\mathcal{A}$ if there is a final state $q \in Q_{\text{fin}}$ such that $d \to q$. The language $\mathcal{L}(\mathcal{A})$ is the set of trees accepted by $\mathcal{A}$.

The class of automata considered in this paper is a subset of a richer (homonym) class of tree automata defined by the authors [16, 17, 24]. In the original version, sheaves automata may be used on terms built from an arbitrary number of free function symbols and from any number of associative and AC operators. Therefore, the definition of sheaves logic may be extended to an arbitrary signature, giving an elegant way to extend our results to an algebra with sequential composition and (not only unary) function symbols. When restricted to tree composition, sheaves automata correspond to a particular instance of *multiset automata* [13], defined by Colcombet to reason on higher-order versions of Process Rewrite Systems. More significantly, we can draw a parallel between sheaves automata and *hedge automata* [25], an extension of regular tree languages at the basis of RELAX-NG [29], a schema language for XML. Whereas hedge automata operate on an ordered model of trees and use regular word languages to constrain ordered bunches (sequences) of elements, we work on an unordered model and use semilinear sets to constrain the multiplicities of unordered bunches (multisets) of elements.

## 6.2 Closure Properties

Given two Sheaves Automata $\mathcal{A} = \langle Q, Q_{\text{fin}}, R\rangle$ and $\mathcal{A}' = \langle Q', Q'_{\text{fin}}, R'\rangle$, we can construct the *product automaton*, $\mathcal{A}\times\mathcal{A}'$, that will prove useful in the definition of the automata for union and intersection. The product $\mathcal{A}\times\mathcal{A}'$ is the automaton $\mathcal{A}^{\times} = \langle Q^{\times}, \emptyset, R^{\times}\rangle$ such that $Q^{\times} = Q\times Q' = \{(q_1,q'_1),\dots,(q_p,q'_r)\}$ and:

- for every type (1) rule $\alpha[q]\to s\in R$ and $\beta[q']\to s'\in R'$, if $\alpha\cap\beta\neq\emptyset$ then the rule $(\alpha\cap\beta)[(q,q')]\to(s,s')$ is in $R^{\times}$,
- for every type (2) rule $\phi\to q\in R$ and $\phi'\to q'\in R'$, the rule $\phi^{\times}\to(q,q')$ is in $R^{\times}$, where $\phi^{\times}$ is the product of the formulas $\phi$ and $\phi'$ obtained as follows. Let $^{\#}(q,q')$ be the variable associated to the numbers of occurrences of the state $(q,q')$, then $\phi^{\times}$ is the formula:

$$\phi\big(\textstyle\sum_{q'\in Q'} {}^{\#}(q_1,q'),\dots,\sum_{q'\in Q'} {}^{\#}(q_p,q')\big)$$
$$\wedge \quad \phi'\big(\textstyle\sum_{q\in Q} {}^{\#}(q,q'_1),\dots,\sum_{q\in Q} {}^{\#}(q,q'_r)\big)$$

The following property states the soundness of this construction.

PROPOSITION 6.1. *We have $d\to(q,q')$ in the automaton $\mathcal{A}\times\mathcal{A}'$, if and only if both $d\to_{\mathcal{A}} q$ and $d\to_{\mathcal{A}'} q'$.*

Given two automata, $\mathcal{A}$ and $\mathcal{A}'$, it is possible to build an automaton accepting the language $\mathcal{L}(\mathcal{A})\cup\mathcal{L}(\mathcal{A}')$ and an automaton accepting $\mathcal{L}(\mathcal{A})\cap\mathcal{L}(\mathcal{A}')$. The intersection $\mathcal{A}\cap\mathcal{A}'$ and the union $\mathcal{A}\cup\mathcal{A}'$ may be obtained from the product $\mathcal{A}\times\mathcal{A}'$ simply by setting the set of final states to:

$$\begin{array}{lcl} Q^{\cap}_{\text{fin}} & =_{\text{def}} & \big\{(q,q') \mid q\in Q_{\text{fin}} \wedge q\in Q'_{\text{fin}}\big\} \\ Q^{\cup}_{\text{fin}} & =_{\text{def}} & \big\{(q,q') \mid q\in Q_{\text{fin}} \vee q\in Q'_{\text{fin}}\big\} \end{array}$$

The union automaton may also be obtained using a simpler construction, similar to the one for finite state (word) automata, leading to an automaton with states $Q\cup Q'$.

PROPOSITION 6.2. *The automaton $\mathcal{A}\cup\mathcal{A}'$ accepts $\mathcal{L}(\mathcal{A})\cup\mathcal{L}(\mathcal{A}')$ and $\mathcal{A}\cap\mathcal{A}'$ accepts $\mathcal{L}(\mathcal{A})\cap\mathcal{L}(\mathcal{A}')$.*

The class of sheaves automata is also closed by complementation. The construction of the complement of an automaton is similar to a determinization procedure. (In particular, the complemented automaton may be exponentially bigger than the original).

PROPOSITION 6.3. *Given an automaton $\mathcal{A}$ we can build an automaton $\mathcal{A}^{\perp}$ such that $\mathcal{L}(\mathcal{A}^{\perp}) = \mathcal{IT}\setminus\mathcal{L}(\mathcal{A})$.*

The product construction yields an efficient algorithm to test the inclusion $\mathcal{L}(\mathcal{A})\subseteq\mathcal{L}(\mathcal{A}')$, provided that $\mathcal{A}'$ is deterministic. In this case, we simply need to test the emptiness of the language accepted by $\mathcal{A}\times\mathcal{A}'$ with final states $Q_{\text{fin}}\times(Q'\setminus Q'_{\text{fin}})$. Using our equivalence and definability results, Th. 5.1 and 6.1, we may relate this problem to testing whether a formula of TL is a "subtype" of another formula, an important issue in the implementation of the programming language TQL.

## 6.3 Membership and Test for Emptiness

In this section, we consider the problem of checking if an information tree is accepted by a given automaton.

Assume there is a function *Cost* such that, for all constraints $\phi$, the evaluation of $\phi(n_1,\dots,n_p)$ can be done in time $O(\mathit{Cost}(p,n))$ whenever $n_i\leqslant n$ for all $i$ in $1..p$. For quantifier-free Presburger formula (and if $n$ is in binary notation) such a function is given by $K.p.\log(K.p.n)$, where $K$ is the greatest coefficient occurring in $\phi$. For arbitrary situations, that is, for formulas with unbounded quantifier alternation, evaluating a formula is as hard as testing its satisfiability and therefore the complexity is triply exponential.

PROPOSITION 6.4. *For an automaton $\mathcal{A} = \langle Q, Q_{fin}, R\rangle$, the model-checking problem, $d\in\mathcal{L}(\mathcal{A})$, can be decided in time $O(|d|.|R|.\mathit{Cost}(|Q|,|d|))$ for a deterministic automaton. The problem is NP-complete for a non-deterministic automaton.*

We give an algorithm for deciding emptiness based on a standard marking algorithm for regular tree automata. The marking algorithm computes two sets of states, $Q_M$ and $Q_E$, where $Q_M$ corresponds to reachable states and $Q_E$ corresponds to states reachable by an element (*i.e.,* through the application of a type (1) rule). The algorithm returns a positive answer if and only if there is a marked final state.

In the case of constrained rules, $\phi\to q$, we need to check whether there is a multiset of marked elements whose mapping satisfies $\phi$. This amounts to checking the satisfiability of the Presburger formula $\phi(^{\#}q_1,\dots,{}^{\#}q_p)\wedge\bigwedge_{q\notin Q_E} {}^{\#}q = 0$. When this formula is satisfiable, we say that the constraint $\phi\setminus Q_E$ is satisfiable. In particular, the constraint $\phi\setminus\emptyset$ is satisfiable if and only if $\models\phi(\mathbf{0})$.

**Algorithm 1. Test for Emptiness**

```
$Q_E = \emptyset$
$Q_M = \{q \mid \phi\to q\in R \wedge \models\phi(\mathbf{0})\}$
**repeat**
    **if** $\alpha[q']\to q\in R$ and $q'\in Q_M$ and $\alpha\neq\emptyset$
    **then** $Q_M := Q_M\cup\{q\}$ and $Q_E := Q_E\cup\{q\}$
    **if** $\phi\to q\in R$ and $\phi\setminus Q_E$ is satisfiable
    **then** $Q_M := Q_M\cup\{q\}$
**until** no new state can be added to $Q_M$
**if** $Q_M$ contains a final state
**then** return *not empty* **else** return *empty*
```

PROPOSITION 6.5. *A state $q$ is marked by Algorithm 1 if and only if there exists a tree $d$ such that $d\to q$.*

We may prove prove this claim using a reasoning similar to the one for regular tree automata. We can also establish a result on the complexity of this algorithm. Let $\mathit{Cost}_{\mathcal{A}}$ denote the maximal time required to decide the satisfiability of the constraints occurring in type (2) rules of $\mathcal{A}$.

PROPOSITION 6.6. *The problem $L(\mathcal{A})\overset{?}{=}\emptyset$ is decidable in time $O(|Q|.|R|.\mathit{Cost}_{\mathcal{A}})$.*

In the case of regular tree automata, it is possible to find a more refined algorithm for the emptiness problem, based on the satisfiability of propositional Horn clauses, with only a linear complexity in the size of $\mathcal{A}$. Therefore we may hope to improve our complexity result.

## 6.4 Results on the Tree Logic

We prove our main property linking sheaves automata and the sheaves logic and use this result to derive several complexity results on the fragment of Ambient Logic studied in this paper.

THEOREM 6.1 (DEFINABILITY). *For every SL-formula $A$, we can build an automaton $\mathcal{A}$ accepting the models of $A$.*

PROOF. By structural induction on the definition of $A$. In the case $A = \top$ we can simply choose an "all accepting" automaton. For example, the automaton with unique (final) state, $q$, and with rules $\mathbb{0}^{\perp}[q] \rightarrow q$, and $(\#q \geqslant 0) \rightarrow q$.

The only other case is $A = \exists \mathbf{N} . \phi . \mathbf{E}$, where $\mathbf{E}$ is a support vector $(\alpha_1[A_1], \dots, \alpha_p[A_p])$. By induction, there is an automaton $\mathcal{A}_i$ accepting the models of $A_i$ for all $i \in 1..p$. From an automaton $\mathcal{C}$ accepting the models of $C$, we can construct an automaton $\mathcal{C}_\alpha$ accepting the set $\{a[d] \mid d \in [\![C]\!], a \in \alpha\}$: if $\alpha = \emptyset$, then $[\![\alpha[C]]\!] = \emptyset$ and we can simply choose for $\mathcal{C}_\alpha$ any automaton with an empty set of final states; otherwise, we obtain $\mathcal{C}_\alpha$ from $\mathcal{C}$ by adding a fresh new state $q_s$, that will be the only final state of $\mathcal{C}_\alpha$, and by adding one type (1) rule $\alpha[q] \rightarrow q_s$ for each final state $q$ of $\mathcal{C}$. Thus, we may build an automaton $\mathcal{B}_i$ accepting the models of $\alpha_i[A_i]$ for all $i \in 1..p$.

The construction of $\mathcal{A}$ is similar to a determinization process. Let $\mathcal{A}$ be the product automaton of the $\mathcal{B}_i$'s and let $\{\mathcal{Q}_1, \dots, \mathcal{Q}_m\}$ be the states of $\mathcal{A}$. A state $\mathcal{Q}$ of $\mathcal{A}$ is of the form $(q_1, \dots, q_p)$, with $q_i$ a state of $\mathcal{B}_i$, and may represent terms accepted by several of the $\mathcal{B}_i$'s. We use the notation $\mathcal{Q} \in \text{fin}(i)$ to say that the $i^{\text{th}}$ component of $\mathcal{Q}$ is a final state of $\mathcal{B}_i$. The constrained rules of $\mathcal{A}$ are of the form $\psi(M_1, \dots, M_m) \rightarrow \mathcal{Q}$, where $M_i$ stands for the number of occurrences of the state $\mathcal{Q}_i$ in a run. The idea is to extend $\mathcal{A}$ with a fresh new state, $q_s$, that will be its only final state, and to add a new rule $\phi^{\exists}(M_1, \dots, M_m) \rightarrow q_s$, where $\phi^{\exists}$ is satisfied by configurations $\mathcal{Q}_{j_1} \mid \cdots \mid \mathcal{Q}_{j_n}$ containing only states in $\text{fin}(i)$ (for some $i \in 1..p$). The formula $\phi^{\exists}$, given below, is obtained by decomposing $M_i$ into a sum of integer variables $X_j^i$, for $j \in 1..p$, corresponding to final states of $\mathcal{B}_j$ occurring in $\mathcal{Q}_i$.

$$\exists (X_j^i)_{\substack{i \in 1..m \\ j \in 1..p}} \cdot \left( \begin{array}{l} \bigwedge_{i \in 1..m} (M_i = \sum_{\substack{j \in 1..p \\ \mathcal{Q}_i \in \text{fin}(j)}} X_j^i) \\ \wedge \ \phi(\sum_{\substack{i \in 1..m \\ \mathcal{Q}_i \in \text{fin}(1)}} X_1^i, \ \dots \ , \sum_{\substack{i \in 1..m \\ \mathcal{Q}_i \in \text{fin}(p)}} X_p^i) \end{array} \right)$$

The property follows by showing that $d \models A$ if and only if $d \in \mathcal{L}(\mathcal{A})$. $\square$

A formula $A$ of SL is basically a (syntax) tree where each node is labeled by a sheaves composition, $\exists \mathbf{N} . \phi . \mathbf{E}$, and has $|\mathbf{E}|$ sons. Using the underlying tree structure of formulas, we can define the height, $h(A)$, and the degree, $d(A)$, of a formula $A$. The construction of the automaton recognizing a formula $A$ requires to compute the product of at most $d(A)$ automata for each composition in $A$. Therefore, the size of the automaton is bounded by $T\hat{\ }(d(A)\hat{\ }h(A))$, where $T$ is a constant bounding the size of the automaton recognizing $\top$. The decidability (and the complexity) of SL follows from Th. 6.1 and Proposition 6.6. The complexity is doubly exponential in the size of $A$ (like in Proposition 6.6, we abstract over the complexity of deciding the satisfiability of the Presburger constraints).

THEOREM 6.2 (SATISFIABILITY). *For any formula $A$ of SL, the satisfiability problem $[\![A]\!] = \emptyset$ is decidable in time $O(T^2\hat{\ }(d(A)\hat{\ }h(A)))$.*

Combined with our previous results on the embedding of TL in SL we also obtain decidability properties for the tree logic, as well as automata-based decision procedures for the model-checking and subtyping problems. Indeed, from a formula $A$ of TL we can build an equivalent SL-formula with the same depth (which is bounded by $|A|$) and with degree at most $3\hat{\ }|A|$. The value 3 comes from the size of the basis used in the encoding of location (Section 5.1). We obtain that the satisfiability problem for TL is in double exponential time. Once more, we abstract over the complexity of Presburger formulas.

THEOREM 6.3 (SATISFIABILITY). *For any formula $A$ of TL, the satisfiability problem $[\![A]\!] = \emptyset$ is decidable in time $O(T^2\hat{\ }(3\hat{\ }|A|^2))$.*

# 7 An Efficient Tree Logic

Constrained rules of sheaves automata, $\phi \rightarrow q$, are used to explore the horizontal structure of trees. Sheaves automata can as easily explore vertical structure and be used to match paths of (nested elements) labels. We can simply take advantage of the intrinsic recursive nature of automata, which may contain cyclic dependencies between states, to compile path expressions.

In this section, we extend the syntax of SL with recursive definitions. For the sake of brevity, we will stay at an informal level compared to the rest of the paper. Actually, our primary goal is to prove that path expressions and iteration are indeed two orthogonal forms of recursion and that our framework can easily be enriched with path expressions. We also study a simple syntactic restriction on formulas that improves the effectiveness of our approach.

**Recursive Sheaves Logic**

| | |
|---|---|
| $X, Y, \dots$ | recursive variables |
| $E ::=$ | element formula |
| $\alpha[X]$ | element with label in $\alpha$ |
| $D ::=$ | recursive definition |
| $X \leftarrow \exists \mathbf{N} . \phi(\mathbf{N}) . \mathbf{E}$ | sheaves composition |
| $A ::=$ | RSL formula |
| $\langle D_1, \dots, D_n ; X \rangle$ | |

The syntax of formulas is even leaner than in SL and is reminiscent of tree grammar: location is restricted to variables and a formula is simply a set of recursive definitions with a distinguished (initial) variable.

The connection with tree grammar is even clearer in the definition of the satisfaction relation. A tree $d$ matches a formula $\langle \mathbf{D}; X \rangle$, denoted $\mathbf{D} \vdash d : X$, if and only if there is a definition $(X \leftarrow \exists \mathbf{n} . \phi . \mathbf{E})$ in $\mathbf{D}$ such that $d \in \mathbf{n} . [\![\mathbf{E}]\!]$ and $\models \phi(\mathbf{n})$, that is, $d \models \exists \mathbf{N} . \phi . \mathbf{E}$ in SL. We also adjust the rule for element formulas. An element $a[d]$ matches $\alpha[Y]$ (in the context $\mathbf{D}$) if and only if $a \in \alpha$ and $\mathbf{D} \vdash d : Y$. For example, if AnyD is the recursive definition $X \leftarrow \exists N . (N \geqslant 0) . \mathbb{0}^{\perp}[X]$ the formula $\langle \text{AnyD}; X \rangle$ matches every tree in $\mathcal{IT}$ (it provides a possible encoding of $\top$).

Granted the relation with tree grammar, it is not necessary to use tree automata to decide the logic. Indeed, there exist efficient ways to manipulate grammars that do not (explicitly) require automata-based techniques. Nonetheless, the compilation from RSL to sheaves automata is straightforward and provides a good idea of how sheaves automata combine well with recursion. An interesting property of the automaton $\mathcal{A}$ obtained from a formula $A$ is that every variable of $A$ corresponds to a single state in $\mathcal{A}$, and the sizes of

$\mathcal{A}$ and $A$ are proportional.

THEOREM 7.1 (DEFINABILITY). *For every formula $A$, we can build an automaton $\mathcal{A}$ of size $O(|A|)$ accepting the models of $A$.*

PROOF. Assume $A = \langle \mathbf{D}; X \rangle$. For every element formula $\alpha[Y]$ and every variable $Z$ occurring in $A$ we set up the states $q_{\alpha[Y]}$ and $q_Z$. Let $Q$ be the set of all such states. Then, for every element formula $\alpha[Y]$ we set up the type (1) rule $\alpha[q_Y] \to q_{\alpha[Y]}$ and for every definition $Y \leftarrow \exists \mathbf{N}.\phi.\mathbf{E}$ in $\mathbf{D}$, with $\mathbf{E} = (\alpha_1[Y_1], \ldots, \alpha_p[Y_p])$, we set up the type (2) rule: $\phi(\#q_{\alpha_1[Y_1]}, \ldots, \#q_{\alpha_p[Y_p]}) \to q_Y$. Let $R$ be the set of all such rules. The sheaves automaton with states $Q$, rules $R$ and final state the singleton $\{q_X\}$ accepts the models of $A$. $\square$

Using the definition of the derived operators given in Section 5.1, we can prove that RSL corresponds to an extension of TL with recursive definitions, where the location operator is limited to recursive variable, $a[X]$. For example, formula (3), below, is satisfied by trees with a path $(a.b)^*$ and (4) is satisfied by trees matching $A$ somewhere.

$$\langle X \leftarrow (a[Y] \mid \top) \vee \mathbf{0}\ , Y \leftarrow (b[X] \mid \top)\ ; X \rangle \tag{3}$$

$$\langle X \leftarrow (\emptyset^{\perp}[X] \mid \top) \vee A\ ; X \rangle \tag{4}$$

This restriction does not excessively limit the expressiveness of the logic. Although the resulting extension of TL is less expressive than the logic enriched with general least and greatest fixpoints, as found in [9], it is possible to encode all the path operators used in TQL [7]. Additionally, this simple syntactical restriction precludes the definition of degenerate recursive formulas, of the form $\mu X.a[X] \mid a[a[X]]$, where variables appear at different depths and match "unbalanced" set of trees (growing as well in breadth and in depth).

A limitation of our approach is that trees must be processed bottom-up. This strategy may be inefficient for large information trees since, in this case, we want to work "on-the-fly", without completely loading a tree before processing it. To avoid this problem, we may impose a simple syntactic restriction on RSL in order to work with top-down sheaves automata. We take inspiration from a restriction on (sequential composition in) XML Schema [30, Section 3.8.6], known as *Consistent Element Declarations*.

We say that a support $(\alpha_1[X_1], \ldots, \alpha_p[X_p])$ has *consistent element declarations* (CED) if every label uniquely determines a recursive variable: for all $i, j \in 1..p$, if $i \neq j$ then $X_i \neq X_j$ and $\alpha_i \cap \alpha_j = \emptyset$. A formula $\langle \mathbf{D}; X \rangle$ has CED if every variable is defined (appears at the left-hand side) exactly once in $\mathbf{D}$ and if the support of every definition in $\mathbf{D}$ has CED. (A formula with consistent element declarations may contain two elements $\alpha[X]$ and $\alpha[Y]$, with $X \neq Y$, provided they do not appear in the same definition.) Once more, this restriction may be transferred to TL. It corresponds to definitions $X \leftarrow A$ such that, for every pair of subformulas $a[X]$ and $a[Y]$ occurring in $A$, we have $X = Y$. Formula (3) is an example of a TL-formula with consistent element declarations.

If we restrict to formulas with consistent element declarations, it is possible to construct top-down sheaves automata accepting the same models. Informally, the construction is based on the fact that, given a definition $X \leftarrow \exists \mathbf{N}.\phi.\mathbf{E}$ to match, the label of an element fully specifies the element formula in $\mathbf{E}$ that we should try to satisfy.

## 8 Conclusion

This paper is concerned with a fragment of ambient logic that may be seen as a kind of regular expression language over tree-like data structures. More formally, it is an algebra with two orthogonal composition operators: *tree composition*, $\_ \mid \_$, that is commutative and follows the horizontal structure of a tree and *location*, $a[\_]$, that is akin to sequential composition $a.\_$ (of a letter with a word) and follows the vertical structure of a tree. In contrast to the situation found with regular tree expressions, there was no equivalent to regular tree automaton for accepting the languages associated to TL.

Our contribution is a class of tree automata for processing information trees and a compilation method that associates to every formula of TL an automaton accepting the same models.

Our approach reveals a connection between TL and arithmetical constraints on vectors of integers, expressed as formulas of Presburger arithmetic. A possible line for future work could be to extend this relation to other examples of substructural logics, like for example additive fragments of Linear Logic or versions of the logic of Bunched Implications [4, 22]. It is worth mentioning that some complexity results on fragments of linear logic have been proven through reduction to problems on Petri Nets [23] (that is, equivalently, on vector addition systems), which may indicate that this relation has already been partially unveiled.

Our automata-based approach to the manipulation of formulas in the ambient logic may be useful in the implementation of query languages based on an unordered tree model, like TQL [7] for example. However, the logic as presented in Section 4, still lacks in-depth recursion, *e.g.* the capacity to encode path expressions, and additional work is needed to formalize the extension with recursive definitions proposed in Section 7. Another line for future work will be to develop a method for obtaining a sheaves automaton directly from a TL-formula. Currently, the construction of a sheaves automaton corresponding to a formula of TL requires to build first an equivalent formula in SL. (In this respect, SL appears as a kind of assembly language.) A benefit of this simplification is that it could lead to a more refined study of the complexity of TL, perhaps enabling us to isolate "simple" classes of queries.

## 9 Acknowledgments

We would like to thank Philip Wadler for pointing out the *Consistent Element Declarations* restriction on XML Schema.

This work is partially supported by ATIP CNRS "Fondements de l'Interrogation des Données Semi-Structurées" and by IST Global Computing PROFUNDIS.

## 10 References

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

[2] B. Boigelot, S. Jodogne, and P. Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In *International Joint Conference on Automated Reasoning (IJCAR)*, volume 2083 of *LNCS*, pp. 611–625. Springer, 2001.

[3] C. Calcagno, L. Cardelli, and A. D. Gordon. Deciding validity in a spatial logic for trees. In *ACM Workshop on Types in Language Design and Implementaion, TLDI '03*, pp. 62–73. ACM Press, 2003.

[4] C. Calcagno, H. Yang, and P. O'Hearn. Computability and complexity results for a spatial assertion language for data

structures. In *Foundations of Software Technology and Theoretical Computer Science (FST & TCS)*, volume 2245 of *LNCS*. Springer, 2001.

[5] L. Cardelli. Describing semistructured data. *SIGMOD Record*, 30(4), 2001. Database Principles Column.

[6] L. Cardelli and L. Caires. A spatial logic for concurrency (part I). In *Theoretical Aspects of Computer Software (TACS)*, volume 2215 of *LNCS*, pp. 1–37. Springer, 2001.

[7] L. Cardelli and G. Ghelli. A query language based on the ambient logic. In *ESOP'01*, volume 2028 of *LNCS*, pp. 1–22. Springer, 2001.

[8] L. Cardelli and A. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 1378 of *LNCS*, pp. 140–155. Springer, 1998.

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

[10] L. Cardelli and A. Gordon. Logical properties of name restriction. In *Typed Lambda Calculus and Applications (TLCA)*, volume 2044 of *LNCS*. Springer, 2001.

[11] W. Charatonik, S. Dal Zilio, A. Gordon, S. Mukhopadhyay, and J.-M. Talbot. Model checking mobile ambients. *Theoretical Computer Science*, 308(1):277–332, 2003.

[12] W. Charatonik and J.-M. Talbot. The decidability of model checking mobile ambients. In *Conference of the European Association for Computer Science Logic (CSL)*, volume 2142 of *LNCS*, pp. 339–354. Springer, 2001.

[13] T. Colcombet. Rewriting in the partial algebra of typed terms modulo AC. In *International Workshop on Verification of Infinite-State Systems (INFINITY)*, volume 68 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 2002.

[14] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997. release October, 1st 2002.

[15] D. Cooper. Theorem-proving in arithmetic without multiplication. *Machine Intelligence*, 7, 1972.

[16] S. Dal Zilio and D. Lugiez. Multitrees automata, Presburger constraints and tree logics. Technical Report 08-2002, LIF, June 2002.

[17] S. Dal Zilio and D. Lugiez. XML schema, tree logic and sheaves automata. In *Rewriting Techniques and Applications (RTA)*, volume 2706 of *LNCS*, pp. 246–263. Springer, 2003. Appears also as Technical Report 4641, INRIA, Nov. 2002.

[18] S. Dal Zilio, D. Lugiez, and C. Meyssonnier. A logic you can count on. Technical report, INRIA, 2003. (to appear).

[19] M. Fischer and M.O.Rabin. Super-exponential complexity of presburger arithmetic. In *SIAM-AMS Symposium in Applied Mathematics*, volume 7, pp. 27–41, 1974.

[20] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax involving binders. In *Symposium on Logic in Computer Science (LICS)*, pp. 214–224. IEEE, 1999.

[21] H. Hosoya and B. C. Pierce. Regular expression pattern matching for XML. In *Principles of Programming Languages (POPL)*, pp. 67–80. ACM Press, 2001.

[22] S. Ishtiaq and P. W. O'Hearn. BI as an assertion language for mutable data structures. In *Principles of Programming Languages (POPL)*, pp. 14–26. ACM Press, 2001.

[23] M. Kanovich. Horn programming in linear logic is NP-complete. In *Symposium on Logic in Computer Science (LICS)*, pp. 200–210. IEEE, 1992.

[24] D. Lugiez. Counting and equality constraints for multitree automata. In *Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 2620 of *LNCS*, pp. 328–342. Springer, 2003.

[25] M. Makoto. Extended path expression for XML. In *Principles of Database Systems (PODS)*. ACM Press, 2001.

[26] D. C. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of presburger arithmetic. *Journal of Computer and System Sciences*, 16:323–332, 1978.

[27] Profundis: Proofs of functionality for mobile distributed systems. `http://www.it.uu.se/profundis/`.

[28] W. Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 35(8):102–114, 1992.

[29] RELAX-NG. `http://www.relaxng.org`.

[30] W3C Recommendation. *XML Schema Part 1: Structures*, 2001.

## A Support Refinements and Bases

In this section, we prove that it is always possible to work with formulas defined over a common basis. To this end, we introduce the notion of *refinement* of a support vector and hint at a method for building a common basis from different supports.

We say that the support **E** refines **F** if from any decomposition (of a tree) over **F** it is possible to extract a more precise decomposition over **E**. We have seen an example of refinement in the encoding of location, $\alpha[A]$. Indeed, we want to say that the basis $(\alpha[A], \alpha[A^\perp], \alpha^\perp[\top])$ is more precise than the support $(\alpha[A])$.

DEFINITION A.1 (REFINEMENT). *A refinement from a support $(E_1, \ldots, E_p)$ into a support $(F_1, \ldots, F_q)$ is a relation $\mathcal{R}$ of $1..p \times 1..q$ such that $[\![E_i]\!] = \bigcup_{(i,j)\in\mathcal{R}} [\![F_j]\!]$ for all $i \in 1..p$.*

We use the notation $\mathbf{F} \preceq_{\mathcal{R}} \mathbf{E}$ when there exists a refinement $\mathcal{R}$ from **E** into **F** and simply say that **F** refines **E** when the relation $\mathcal{R}$ is obvious from the context. We prove that if **F** refines **E** then any SL-formula with support **E** can be rewritten into an equivalent formula with support **F**.

PROPOSITION A.1. *Assume $\mathbf{E} = (E_1, \ldots, E_p)$ and $\mathbf{F} = (F_1, \ldots, F_q)$ are two support vectors such that $\mathbf{F} \preceq_{\mathcal{R}} \mathbf{E}$, then for every counting constraint $\phi(\mathbf{N})$, with $|\mathbf{N}| = p$, we have:*

$$[\![\exists \mathbf{N} . \phi(\mathbf{N}) . \mathbf{E}]\!] = [\![\exists \mathbf{M} . \varphi_{\mathcal{R}}(\mathbf{M}) . \mathbf{F}]\!] \; .$$

*where $\mathbf{M} = (M_1, \ldots, M_q)$and $\varphi_{\mathcal{R}}(\mathbf{M})$ is the constraint:*

$$\exists (X_j^i)_{(i,j)\in\mathcal{R}} . \left( \begin{array}{l} \bigwedge_{j\in 1..q} (M_j = \sum_{(i,j)\in\mathcal{R}} X_j^i) \\ \wedge\, \phi(\sum_{(1,j)\in\mathcal{R}} X_j^1, \ldots, \sum_{(p,j)\in\mathcal{R}} X_j^p) \end{array} \right) .$$

PROOF. We prove $[\![\exists \mathbf{N} . \phi . \mathbf{E}]\!] \subseteq [\![\exists \mathbf{M} . \varphi_{\mathcal{R}} . \mathbf{F}]\!]$. The proof for the other direction is similar. Assume $d \in [\![\exists \mathbf{N} . \phi . \mathbf{E}]\!]$. Then $d$ is the composition of $n_1$ elements satisfying $E_1$ (say $e^1_1, \ldots, e^1_{n_1}$), ..., and $n_p$ elements satisfying $E_p$ (say $e^p_1, \ldots, e^p_{n_p}$) with $\models \phi(\mathbf{n})$.

By definition of $\mathbf{F} \preceq_{\mathcal{R}} \mathbf{E}$, we have $[\![E_i]\!] = \bigcup_{i \mathcal{R} j} [\![F_j]\!]$. Therefore every element $e^i_j$ in this decomposition should also fall into one of the components of $\mathbf{F}$, that is, there exists an index $r_{i,j}$ in $1..q$ such that $i \,\mathcal{R}\, r_{i,j}$ and $e^i_j \in [\![F_{r_{i,j}}]\!]$ for all $i \in 1..p, j \in 1..n_i$. This gives a new decomposition of $d$ into the support $\mathbf{F}$ such that the cardinality of $\{r_{i,j} = k \mid i \in 1..p, j \in 1..n_i\}$, say $m_k$, is the number of elements matching $F_k$. The relation $d \in [\![\exists \mathbf{M}. \varphi_{\mathcal{R}} . \mathbf{F}]\!]$ follows by proving that $\varphi_{\mathcal{R}}(m_1, \ldots, m_q)$ holds.

Let $x^i_k$ be the cardinality of the set $\{r_{i,j} = k \mid j \in 1..n_i\}$, that is, the number of elements $e^i_j$, for $j \in 1..n_i$, assigned to $F_k$. Therefore $n_i = \sum_{i \mathcal{R} k, k \in 1..q} x^i_k$ for all $i \in 1..p$, which implies $\models \phi(\sum_{1 \mathcal{R} j} x^1_j, \ldots, \sum_{p \mathcal{R} j} x^p_j)$, and $m_k = \sum_{i \mathcal{R} k, i \in 1..p} x^i_k$ for all $k \in 1..q$, as required. $\Box$

Given two support vectors $\mathbf{E}$ and $\mathbf{F}$, a corollary of Proposition A.1 is that from every formula $\exists \mathbf{N}. \phi . \mathbf{E}$ we can build an equivalent formula of the form $\exists(\mathbf{N};\mathbf{M}). \varphi . (\mathbf{E};\mathbf{F})$, where $\varphi(\mathbf{N};\mathbf{M})$ is the constraint $\phi(\mathbf{N}) \wedge (\mathbf{M} = \mathbf{0})$. This property follows from the fact that the support $(\mathbf{E};\mathbf{F})$ clearly refines $\mathbf{E}$. Hence, we may always work with formulas using a common support (from formulas with support $\mathbf{E_1}, \ldots, \mathbf{E_n}$, use the support $\mathbf{E_1}; \ldots; \mathbf{E_n}$). This property is useful when we need to build the disjunction or parallel composition of arbitrary formulas.

We need a stronger property for the "negative" operators: negation and composition adjunct.

PROPOSITION A.2. *Assume* $\mathbf{E}$ *and* $\mathbf{F}$ *are two proper bases then we can build a new basis, say* $\mathbf{E} \times \mathbf{F}$*, that refines both* $\mathbf{E}$ *and* $\mathbf{F}$*.*

PROOF. By induction on the depth of $\mathbf{E}$. Assume $\mathbf{E} = (E_1, \ldots, E_p)$ and $\mathbf{F} = (F_1, \ldots, F_q)$, where $E_i = \alpha_i[A_i]$ and $F_j = \beta_j[B_j]$ for all $i \in 1..p, j \in 1..q$.

We choose for $\mathbf{E} \times \mathbf{F}$ the support vector with element formulas $G_{i,j} = (\alpha_i \cap \beta_j)[A_i \wedge B_j]$, for all $i \in 1..p, j \in 1..q$, where $A_i \wedge B_j$ is defined as follows.

If $A_i = \top$ then $A_i \wedge B_j = B_j$. If $B_j = \top$ then $A_i \wedge B_j = A_i$. Otherwise, the formulas $A_i$ and $B_j$ are compositions defined over proper bases of lesser depths. Let $\mathbf{C}_i$ (*resp.* $\mathbf{D}_j$) be the basis used in the definition of $A_i$ (*resp.* $B_j$). By induction hypothesis there is a proper basis $\mathbf{C_i} \times \mathbf{D_j}$ refining $\mathbf{C}_i$ and $\mathbf{D}_j$. Hence, by Proposition A.1, we may rewrite $A_i$ and $B_j$ into equivalent formulas defined over the basis $\mathbf{C}_i \times \mathbf{D}_j$ and we can build a formula equivalent to $A_i \wedge B_j$ using the derived operator defined in Section 5.1.

The proposition follows by proving that $\mathbf{G}$ is a proper basis refining $\mathbf{E}$ and $\mathbf{F}$. We have $[\![G_{i,j}]\!] = [\![E_i]\!] \cap [\![F_j]\!]$. Therefore $[\![G_{i,j}]\!] \cap [\![G_{k,l}]\!] = ([\![E_i]\!] \cap [\![E_k]\!]) \cap ([\![F_j]\!] \cap [\![F_l]\!])$ and we have $[\![G_{i,j}]\!] \cap [\![G_{k,l}]\!] = \emptyset$ if and only if $i = k$ and $j = l$. Hence, the formulas in $\mathbf{G}$ are linearly independent. Moreover, $\bigcup_{i,j} [\![G_{i,j}]\!] = \bigcup_i [\![E_i]\!] = \mathcal{E}$, that is, $\mathbf{G}$ is generating. Finally, the relations $\mathcal{R}$ and $\mathcal{R}'$ such that $i \,\mathcal{R}\, (i,j)$ and $j \,\mathcal{R}'\, (i,j)$ for all $i \in 1..p, j \in 1..q$ are refinements from $\mathbf{E}$ and $\mathbf{F}$ to $\mathbf{G}$, as needed. $\Box$

The support obtained by this operation may be further simplified by eliminating useless components, namely formulas of the form $\emptyset[A]$ obtained from the conjunction of elements with disjoint label expressions.

We show that it is always possible to assume that a set of formulas is defined over a common proper basis. Consequently, we may extend the encodings given in Section 5.1 to arbitrary formulas of SL.

PROPOSITION A.3 (SUPPORT REFINEMENT). *Given a sequence of formulas* $(A_1, \ldots, A_n)$*, where* $A_i$ *is defined over a proper basis for all* $i \in 1..n$*, we can build a sequence of formulas* $(B_1, \ldots, B_n)$*, defined on a common proper basis, such that* $[\![A_i]\!] = [\![B_i]\!]$ *for all* $i \in 1..n$*.*

PROOF. By Propositions A.1 and A.2 using the product vector $(\mathbf{E}_1 \times \ldots) \times \mathbf{E}_n$, where $(\mathbf{E}_i)_{i \in 1..n}$ are the proper basis used in the definition of the formulas $(A_i)_{i \in 1..n}$. $\Box$

We can use a similar technique to prove that, from any support $\mathbf{E}$, we may always obtain a proper basis refining $\mathbf{E}$. The idea is to study element formulas $B_I$ equivalent to $\bigwedge_{i \in I} E_i \wedge \bigwedge_{i \notin I} \neg E_i$, where $I$ is a subset of $1..p$. The proof of this result is slightly more involved than the proof of Proposition A.2 since, in the general case, $\neg(\alpha[A])$ is not equivalent to an element formula, but rather to the disjunction of $\alpha^{\perp}[\top]$ and $\alpha[\neg A]$. This stronger property is not needed in the proof of our main result. Moreover, whereas the product of two bases may generate a vector of size at most quadratic, the more sophisticated construction may generate an exponential number of element formulas.