A Logic You Can Count On

Silvano Dal Zilio, Denis Lugiez, Charles Meyssonnier

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Logic You Can Count On

Silvano Dal Zilio — Denis Lugiez — Charles Meyssonnier

N° 5022

November 2003

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--5022--FR+ENG

A Logic You Can Count On

Silvano Dal Zilio, Denis Lugiez, Charles Meyssonnier*

Thème 1 — Réseaux et systèmes
Projet Mimosa

Rapport de recherche n° 5022 — November 2003 — 33 pages

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.

Key-words: Tree automata, spatial logics, semi-structured data.

* The authors work at the *Laboratoire d'Informatique Fondamentale de Marseille*, UMR 6166, CNRS and Université de Provence, and are partly supported by ATIP "Fondements de l'Interrogation des Données Semi-Structurées" and IST Profundis. A portion of this work appears in *Proceedings of the 31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (POPL), ACM Press, 2004.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 —Télécopie : +33 4 92 38 77 65

Vous pouvez compter sur cette logique

Résumé : Nous prouvons la décidabilité du fragment statique, sans quantificateurs, de la logique des ambients avec l'adjoint de la composition et l'itération, qui est l'équivalent d'un langage d'expressions régulières pour les données semi-structurées. L'essence de ce résultat est une connexion surprenante entre les formules de la logiques des ambients et des contraintes arithmétiques sur des vecteurs d'entiers.

Notre technique de preuve est fondée sur une nouvelle classe d'automates d'arbres et peut servir de base à l'implémentation d'algorithmes pour décider la satisfiabilité d'une formule. Un des bénéfices de notre approche est de se prêter naturellement à une extension de la logique par des définitions récursives, qui est elle aussi décidable. Enfin, nous identifions une restriction syntaxique simple qui améliore l'efficacité de notre approche lorsqu'on travaille sur de grands exemples.

Mots-clés : Automate d'arbres, logique spatiale, données semi-structurées.

He who goes out weeping, carrying seeds to sow, will return with songs of joy, carrying sheaves with him.

126:6

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 this fragment 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 [28], 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 π-calculus [6] or almost every calculus with a system of nested locations [11, 19]). 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 [12] and that it is undecidable for the logic with name quantification and composition adjunct [13]. In [16], the authors show decidability of the satisfiability problem for the logic without adjunct and name quantification using tree automata. 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. We also show that the satisfiability problem is in time elementary. Our approach 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 TL.

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 [15]. 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, 3 and 4 we review background material on information trees, on the Tree Logic (TL) and on Presburger arithmetic. In Section 5 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.

Then 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 7) 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 [31], 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 5), with a study of the algorithmic complexity of our methods and the definition of possible simplifying restrictions.

2 Information Trees

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 Λ 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
$\quad a[d]$	element labeled a (with $a \in \Lambda$), containing d
$d ::=$	Information tree
$\quad \mathbf{0}$	empty information tree
$\quad e$	element
$\quad d \mid d'$	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 2.1 *The following information tree*[1] *may be interpreted as a valid entry for the bibliographical reference [8]:*

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

2.1 Notations

In the following, it will often be convenient to use *label expressions* to represent finite or cofinite sets of labels. Label expressions, ranged over by $\alpha, \beta, \ldots$, are either of the shape $a_1, \ldots, a_n$, for the finite set of labels $\{a_1, \ldots, a_n\} \subseteq \Lambda$, or of the shape $\alpha^\perp$, for the complement of the set denoted by the label expression α. We shall write $a \in \alpha$ when either $\alpha = a_1, \ldots, a_n$ and $a = a_i$ for some $i \in 1..n$, or $\alpha = \beta^\perp$ and $a \notin \beta$.

We also extend composition to sets of information trees, using the notation $S \mid S'$ to denote the set $\{(d \mid d') \mid (d, d') \in S \times S'\}$. It is easy to check that composition on sets of trees is distributive over set union, i.e.: $S \mid (S_1 \cup S_2) = (S \mid S_1) \cup (S \mid S_2)$. When S is a set of trees, we write S^0 for the singleton $\{\mathbf{0}\}$, S^n for the set $S \mid \ldots \mid S$ (n times), and S^* for the set $\bigcup_{n \in \mathbb{N}} S^n$.

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 $\mathbf{n}.\mathbf{S}$ 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 report.

3 Tree Logic

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, also called the Tree Logic (TL), only refers to the spatial distribution of locations and appears particularly well-suited to describe the structure of information trees. In this paper, we study the quantifier-free fragment of 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 (breadth-)recursive formulas.

Tree Logic Syntax

$A, B ::=$	formula
$\top$	true
$\neg A$	negation
$A \vee B$	disjunction
$\mathbf{0}$	empty tree

[1] In order to ease the readability of our examples we use different fonts for the encoding of string constants.

$\alpha[A]$	location
$A \mid B$	composition
$A \rhd B$	composition adjunct
A^*	indefinite repetition

Another difference with to the original presentation of TL as found in [7] is the use of label expressions (see Sect.2.1) in the definition of the location construct. This small extension compensates for the absence of an operator for existential quantification over labels, $\exists x.A$, in our syntax. Our motivation for this extension is that, although the addition of "full quantification" breaks our decidability results, it is workable to consider location formulas over *any* possible label, which can be written $\exists x.x[A]$ with a quantifier (where x does not appear free in A), or $\emptyset^{\perp}[A]$ with a label expression.

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 $d \in [\![A]\!]$.

Tree Logic Semantics

$$
\begin{array}{lcl}
[\![\top]\!] & = & \mathcal{IT} \\
[\![\neg A]\!] & = & \mathcal{IT} \setminus [\![A]\!] \\
[\![A \vee B]\!] & = & [\![A]\!] \cup [\![B]\!] \\
[\![\mathbf{0}]\!] & = & \{\mathbf{0}\} \\
[\![\alpha[A]]\!] & = & \{a[d] \mid a \in \alpha \ , \ d \in [\![A]\!]\} \\
[\![A \mid B]\!] & = & [\![A]\!] \mid [\![B]\!] \\
[\![A \rhd B]\!] & = & \{d \mid \forall d' \in [\![A]\!] \ . \ (d \mid d') \in [\![B]\!]\} \\
[\![A^*]\!] & = & [\![A]\!]^*
\end{array}
$$

The satisfaction rules for the propositional fragment are conventional. The formula $\mathbf{0}$ only matches (trees structurally congruent to) $\mathbf{0}$; location $\alpha[A]$ matches trees with a single branch labeled a at the root, with $a \in \alpha$, leading to a subtree satisfying A; composition $A_1 \mid A_2$ matches all compositions of one tree satisfying A_1 with one tree satisfying A_2. Finally, composition adjunct $A \rhd B$ matches d if and only if for all trees d' that satisfy A, the composition $d \mid d'$ satisfies B, and indefinite repetition A^* matches compositions of an arbitrary number of trees satisfying A.

Example 3.1 *The following formula matches bibliographical entries corresponding to papers written by* `Cardelli` *in* `1998`*, and in particular the information tree given in Example 2.1:*

$$\mathit{article}[\mathit{title}[\top] \mid \mathit{author}[\texttt{Cardelli}[]] \mid \mathit{year}[\texttt{1998}[]] \mid \neg((\mathit{title}, \mathit{year})[\top] \mid \top)] \ .$$

This formula specifies that the article must have a title, the author `Cardelli`*, and the year of publication* `1998`*. The remaining part of the formula,* $\neg((\mathit{title}, \mathit{year})[\top]) \mid \top)$*, specifies that a valid entry may also contain other fields (for instance the journal in which the paper was published), provided that none of these fields matches* *title* *or* *year*.

On the same line of thought, we could define a validity criterion for bibliographical entries as the formula:

$$article[title[\top] \mid author[\top] \mid \neg((title[\top] \vee (year[\top] \mid year[\top])) \mid \top)] \ . \tag{1}$$

Formula (1) 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 with some other (unspecified) fields. These constraints could be expressed more directly using the sheaved composition notation of section 2.1, by saying that a valid entry must be in one of the sets:

$$(n_t, n_a, n_y, n_o) \cdot ([\![title[\top]]\!], [\![author[\top]]\!], [\![year[\top]]\!], [\![(title, author, year)^{\perp}[\top]]\!]) \ ,$$

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 $(n_o \geqslant 0)$. One of the main contributions of this report 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 (that is, formulas of the kind $\alpha[A]$).

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.

4 Presburger Arithmetic

Presburger arithmetic is the first-order theory of equality over the group $(\mathbb{N}, +)$ of natural numbers with addition [27]. Presburger formulas (also called constraints) are described in the following table, where $M, N, \ldots$ range over positive integer variables and $m, n, \ldots$ range over positive integer constants.

Presburger Constraints

$Exp ::=$	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 report, 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 use the notations $\phi(\mathbf{N})$ for a Presburger formula whose free variables are all in $\mathbf{N} = (N_1, \ldots, N_p)$ and $\models \phi(n_1, \ldots, n_p)$ 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 [18]: every algorithm which decides the truth of a Presburger constraint ϕ, 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 ϕ. 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 ϕ is in time at most $2\hat{}(2\hat{}(2\hat{}(pn)))$. The problem is NP-complete for the existential fragment of Presburger arithmetic.

4.1 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 the 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 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. Semilinear sets are closed under addition (with $L + M =_{\text{def}} \{x + y \mid x \in L, y \in M\}$), under basic set operations (union, intersection, set complement, ...) and under *iteration* (with $L^n =_{\text{def}} L + \cdots + L$, n times, and $L^* =_{\text{def}} \bigcup_{n \in \mathbb{N}} L^n$). In the case of iteration, the semilinear set L^* may be a union of exponentially many linear sets (in the number of linear sets in L).

Proposition 4.1 (From [16]) *For any two semilinear sets L, M of $\mathbb{N}^p$, the sets $L + M$, L^k (for any $k \in \mathbb{N}$) and L^* are also semilinear sets of $\mathbb{N}^p$.*

Presburger constraints and semilinear sets are *effectively* equivalent, that is, given a Presburger constraint ϕ, it is possible to compute the bases and periods of a semilinear set representing $[\![\phi]\!]$, and conversely. To build a semilinear set corresponding to a given Presburger constraint, it is enough to perform quantifier elimination on the constraint [27], a possibly expensive procedure.

4.2 Derived Operators

Building upon the connection between semilinear sets and Presburger arithmetic, we can lift the sum and iteration operators on semilinear sets to the level of the logic, that is, we can derive connectors $+$ and $*$ such that $[\![\phi + \psi]\!] = [\![\phi]\!] + [\![\psi]\!]$, and $[\![\phi^*]\!] = [\![\phi]\!]^*$. In the following, we make use of the usual derived connectives, *conjunction* $\wedge$, *implication* $\rightarrow$, and *universal quantification* $\forall$.

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

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

It will also prove useful to define, as in TL, the adjunct of sum, that is a connector $\triangleright$ such that $\xi \vdash \phi \triangleright \psi$ if and only if $\xi + \phi \vdash \psi$ (where the entailment relation, $\phi \vdash \psi$, means that $[\![\phi]\!] \subseteq [\![\psi]\!]$). The following definition is reminiscent of the correspondence between $\triangleright$ and the linear implication connector of linear logic, which was already mentioned in [9].

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

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

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

Proposition 4.2 *Assume ϕ and ψ are two Presburger formulas, then for every set S of integer vectors, we have:*

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

Proof For the first equivalence, we prove that $[\![\phi]\!] + [\![\psi]\!] = [\![\phi + \psi]\!]$. Assume $\mathbf{n}$ is a vector in $\mathbb{N}^p$, we have $\mathbf{n} \in [\![\phi]\!] + [\![\psi]\!]$ if and only if there exist two vectors $\mathbf{n}_1$ and $\mathbf{n}_2$ such that $\mathbf{n} = \mathbf{n}_1 + \mathbf{n}_2$ and $\mathbf{n}_1 \in [\![\phi]\!]$ and $\mathbf{n}_2 \in [\![\psi]\!]$, that is, if and only if $\mathbf{n} \in [\![\phi + \psi]\!]$.

For the second equivalence, it is enough to prove that $\mathbf{n}$ in $[\![\phi \triangleright \psi]\!]$ if and only if $\mathbf{n} + [\![\phi]\!] \subseteq [\![\psi]\!]$. By definition of $\triangleright$, we have $\mathbf{n}$ in $[\![\phi \triangleright \psi]\!]$ if and only if for all $\mathbf{n}_1 \in [\![\phi]\!]$, there is $\mathbf{n}_2$ in $[\![\psi]\!]$ such that $\mathbf{n} + \mathbf{n}_1 = \mathbf{n}_2$ that is, $\mathbf{n} + \mathbf{n}_1 \in [\![\psi]\!]$, as needed. ■

It is also possible to derive a formula for iteration, ϕ^*, 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 ϕ. We take the empty sum to stand for the null vector, $\mathbf{0} = (0, \ldots, 0)$. The construction is quite complex, and requires in the first place to compute the bases and periods of the semilinear set $[\![\phi]\!]$.

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

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

Intuitively, the formula $\phi^*(\mathbf{n})$ constraints the vector $\mathbf{n}$ to be the sum of elements taken from $\bigcup_{i\in 1..l} L(\mathbf{b}_i, P_i)$, where an element in $L(\mathbf{b}_i, P_i)$ (for all $i \in 1..l$) is by definition a solution of ϕ.

Proposition 4.3 *Assume ϕ is a Presburger formula with p free variables, then $\models \phi^*(\mathbf{n})$ if and only if there exists a finite sequence $(\mathbf{n}_1, \ldots, \mathbf{n}_k)$ of elements in $\mathbb{N}^p$ such that $\mathbf{n} = \sum_{i\in 1..k}\mathbf{n}_i$ and $\models \phi(\mathbf{n}_i)$ for all $i \in 1..k$.*

Proof Let ϕ be a Presburger formula denoted by the semilinear set $\bigcup_{i\in 1..l} L(\mathbf{b}_i, P_i)$, with $P_i = \{\mathbf{p}_{i,1}, \ldots, \mathbf{p}_{i,l_i}\}$ for all $i \in 1..l$.

Assume $\models \phi^*(\mathbf{n})$. There are integers μ_i (for $i \in 1..l$) and $\lambda_{i,j}$ (for $i \in 1..l$ and $j \in 1..l_i$) such that $\mathbf{n} = \sum_{i\leqslant l}\big(\mu_i\mathbf{b}_i + \sum_{j\leqslant l_i}\lambda_{i,j}\mathbf{p}_{i,j}\big)$ and $\big(\bigvee_{j\leqslant l_i}\lambda_{i,j}\neq 0\big) \rightarrow \mu_i \neq 0$ for all $i \in 1..l$. Letting $I = \{i \in 1..l \mid \mu_i \neq 0\}$, we have $\mu_i = \lambda_{i,j} = 0$ for all $i \in 1..l \setminus I$, $j \in 1..l_i$, and thus $\mathbf{n} = \sum_{i\in I}\big(\mu_i\mathbf{b}_i + \sum_{j\leqslant l_i}\lambda_{i,j}\mathbf{p}_{i,j}\big)$. For $i \in I$, let $\mathbf{n}_{i,1} = \mathbf{b}_i + \sum_{j\leqslant l_i}\lambda_{i,j}\mathbf{p}_{i,j}$, and, if $\mu_i \geqslant 2$, let $\mathbf{n}_{i,2} = \ldots = \mathbf{n}_{i,\mu_i} = \mathbf{b}_i$. For all $i \in I$, and for all $j \in 1..\mu_i$, we have $\mathbf{n}_{i,j} \in L(\mathbf{b}_i, P_i)$, and thus $\models \phi(\mathbf{n}_{i,j})$, while $\mathbf{n} = \sum_{i\in I}\big(\mu_i\mathbf{b}_i + \sum_{j\leqslant l_i}\lambda_{i,j}\mathbf{p}_{i,j}\big) = \sum_{i\in I}\big(\sum_{j\leqslant\mu_i}\mathbf{n}_{i,j}\big)$.

For the converse, assume there are $\mathbf{n}_1, \ldots, \mathbf{n}_k$ such that $\mathbf{n} = \sum_{h\in 1..k}\mathbf{n}_h$ and $\models \phi(\mathbf{n}_h)$ for all $h \in 1..k$. For each $h \in 1..k$, $\mathbf{n}_h$ must fall into at least one of the $L(\mathbf{b}_i, P_i)$ for $i \in 1..l$, so there is a sequence of integers $(\mu_i)_{i\in 1..l}$ such that we can decompose the sum $\sum_{h\in 1..k}\mathbf{n}_h$ into $\mathbf{n} = \sum_{i\leqslant l}\big(\sum_{h\leqslant\mu_i}\mathbf{n}_{i,h}\big)$, where $\mathbf{n}_{i,h} \in L(\mathbf{b}_i, P_i)$ for all $i \in 1..l$, $h \in 1..\mu_i$. Thus, there are integers $\lambda_{i,h,j}$, for $i \in 1..l$, $h \in 1..\mu_i$ and $j \in 1..l_i$, such that $\mathbf{n}_{i,h} = \mathbf{b}_i + \sum_{j\leqslant l_i}\lambda_{i,h,j}\mathbf{p}_{i,j}$, yielding $\mathbf{n} = \sum_{i\leqslant l}\big(\sum_{h\leqslant\mu_i}\big(\mathbf{b}_i + \sum_{j\leqslant l_i}\lambda_{i,h,j}\mathbf{p}_{i,j}\big)\big)$, that is $\mathbf{n} = \sum_{i\leqslant l}\big(\mu_i\mathbf{b}_i + \sum_{j\leqslant l_i}\lambda_{i,j}\mathbf{p}_{i,j}\big)$, with $\lambda_{i,j} = \sum_{h\leqslant\mu_i}\lambda_{i,h,j}$. We conclude simply by noticing that for all $i \in 1..l$, if $\mu_i = 0$ then $\lambda_{i,j} = 0$ for all $j \in 1..l_i$. ■

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 sum and negation may lead to Presburger formulas with an unbounded alternation of quantifiers, the use of $\triangleright$ does not appear 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.

5 Sheaves Logic

The Sheaves Logic (SL) is a new modal logic for information trees that directly encompasses Presburger constraints, and is based on the "sheaved composition" notation of Section 2.1.

Sheaves Logic Syntax

$E ::=$	Element formula
$\alpha[A]$	element with label in α
$A ::=$	Counting formula
$\top$	true
$\exists \mathbf{N}.\, \phi\, .\, \mathbf{E}$	sheaved composition (with $\lvert\mathbf{N}\rvert = \lvert\mathbf{E}\rvert$)

We again use the letters $A, B, \ldots$ to range over formulas of SL, but this should not cause any ambiguity with TL-formulas. We refer to the sequence $\mathbf{E}$ of element formulas appearing in a sheaved 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 $a \in \alpha$ and $d \models A$. The 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 ϕ. The formula $\top$ does not constrain its models in any way.

Sheaves Logic Semantics

$[\![\alpha[A]]\!]$	$=$	$\{a[d] \mid a \in \alpha\ ,\ d \in [\![A]\!]\}$
$[\![(E_1, \ldots, E_p)]\!]$	$=$	$([\![E_1]\!], \ldots, [\![E_p]\!])$
$[\![\top]\!]$	$=$	$\mathcal{IT}$
$[\![\exists \mathbf{N}.\, \phi\, .\, \mathbf{E}]\!]$	$=$	$\bigcup_{\mathbf{n} \in [\![\phi]\!]} \mathbf{n}\, .\, [\![\mathbf{E}]\!]$

The semantic definition of sheaved composition is probably easier to understand in terms of the associated satisfaction relation. Assume $\mathbf{E}$ is the support $(E_1, \ldots, E_p)$ and $\mathbf{n} = (n_1, \ldots, n_p)$. An information tree is in the set $\mathbf{n}\, .\, [\![\mathbf{E}]\!]$ 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.

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

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

5.1 Bases

The syntax of SL does not restrict the 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, $\bigcup_{\mathbf{n} \in \mathbb{N}^p} \mathbf{n}\, .\, [\![\mathbf{E}]\!] = \mathcal{IT}$ (or equivalently $\bigcup_{i \in 1..p} [\![E_i]\!] = \mathcal{E}$). Another interesting property of support is to have element formulas with disjoint denotations. In the latter case, we say that the element formulas are *linearly independent*. In this case, the decomposition of a tree is always unique, that is, if $\mathbf{n} \neq \mathbf{m}$ then $\mathbf{n}.[\![\mathbf{E}]\!] \cap \mathbf{m}.[\![\mathbf{E}]\!] = \emptyset$. Drawing a parallel with linear algebra, we define a notion of "good" support vectors, that we call *bases*, which are generating sequences of linearly independent element formulas.

Definition 5.1 (Basis) *A vector $(E_1, \ldots, E_p)$ 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 a basis is the singleton sequence AnyE. Another simple example is the sequence $(a_1[\top], \ldots, a_p[\top], \Sigma^{\perp}[\top])$, where $\Sigma = \{a_1, \ldots, a_p\}$ is a finite subset of Λ.

The next proposition states the fundamental property of bases, which is that any information tree admits a unique decomposition following a given basis.

Proposition 5.1 *Assume $\mathbf{E} = (E_1, \ldots, E_p)$ is a basis, then for any information tree, d, there is a unique integers vector, $\mathbf{n} \in \mathbb{N}^p$, such that $d \in \mathbf{n} \,.\, [\![\mathbf{E}]\!]$.*

Proof Let d be the composition $e_1 \mid \ldots \mid e_l$, with $e_i \in \mathcal{E}$ for all $i \in 1..l$. Since E is a basis, an element e_i in the decomposition of d belongs to exactly one set $[\![E_k]\!]$. Let n_k denotes the number of elements in d that belongs to $[\![E_k]\!]$. By construction, we have $d \in \mathbf{n} \,.\, [\![\mathbf{E}]\!]$ and this is the only way to decompose d following $\mathbf{E}$. ■

5.2 Refinements

We introduce the notion of *refinement* of a support vector and describe a method for building a common basis from heterogeneous supports.

Informally, the support $\mathbf{F}$ is a refinement of $\mathbf{E}$ if from any decomposition of a tree over $\mathbf{E}$ it is possible to extract a more precise decomposition over $\mathbf{F}$. For example, the support $(\alpha[A], \alpha[\neg A])$ is a refinement of $(\alpha[\top])$.

Definition 5.2 (Refinement) *A refinement from the support $\mathbf{E} = (E_1, \ldots, E_p)$ into the support $\mathbf{F} = (F_1, \ldots, F_q)$ is a relation $\mathcal{R}$ of $1..p \times 1..q$ such that for all $i \in 1..p$ we have $[\![E_i]\!] = \bigcup_{(i,j) \in \mathcal{R}} [\![F_j]\!]$.*

We use the notation $\mathbf{F} \preceq_{\mathcal{R}} \mathbf{E}$ when there exists a refinement $\mathcal{R}$ from $\mathbf{E}$ into $\mathbf{F}$ and simply write $\mathbf{F} \preceq \mathbf{E}$ when the relation $\mathcal{R}$ is obvious from the context. The main property of refinements is that when $\mathbf{F}$ refines $\mathbf{E}$, any SL-formula with support $\mathbf{E}$ can be rewritten into an equivalent formula with support $\mathbf{F}$.

Proposition 5.2 *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 any Presburger constraint $\phi(\mathbf{N})$, with $\mathbf{N} = (N_1, \ldots, N_p)$, we have :*

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

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

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

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 m_k of $\{(i,j) \in 1..p \times 1..n_i \mid r_{i,j} = k\}$ 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 $\{j \in 1..n_i \mid r_{i,j} = k\}$, that is, the number of elements e^i_j, for $j \in 1..n_i$, assigned to F_k. We have $n_i = \sum_{i\mathcal{R}k, k\in 1..q} x^i_k$ 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. ■

Proposition 5.3 *Assume* $\mathbf{E}$ *and* $\mathbf{F}$ *are two proper bases. We can build a proper basis, denoted* $\mathbf{E} \times \mathbf{F}$*, 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 proceed by case analysis on the formulas $(A_i)_{i\in 1..p}$.

If $A_i = \top$ for all $i \in 1..p$, let $\mathbf{G}$ be the support with element formulas $G_{i,j} = (\alpha_i \cap \beta_j)[B_j]$ for all $i \in 1..p, j \in 1..q$. We prove that $\mathbf{G}$ is a basis refining $\mathbf{E}$ and $\mathbf{F}$.

We have $[\![G_{i,j}]\!] = [\![E_i]\!] \cap [\![F_j]\!]$. Since $[\![G_{i,j}]\!] \cap [\![G_{k,l}]\!] = ([\![E_i]\!] \cap [\![E_k]\!]) \cap ([\![F_j]\!] \cap [\![F_l]\!])$, 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.

Otherwise, there is $n \geqslant 1$ such that for all $i \in 1..p$, A_i is of depth at most n. Let $\mathbf{C}_i$ (*resp.* $\mathbf{D}_j$) be the basis used in the definition of A_i (*resp.* B_j). By induction hypothesis we have that for all $(i,j) \in 1..p \times 1..q$, there is a proper basis $\mathbf{C}_i \times \mathbf{D}_j$ refining $\mathbf{C}_i$ and $\mathbf{D}_j$. By Proposition 5.2, we may rewrite A_i and B_j into equivalent formulas defined over the basis $\mathbf{C}_i \times \mathbf{D}_j$. Therefore, we can build a formula equivalent to $A_i \wedge B_j$. The proposition follows by choosing for $\mathbf{E} \times \mathbf{F}$ the support vector with element formulas $(\alpha_i \cap \beta_j)[A_i \wedge B_j]$ for all $i \in 1..p, j \in 1..q$. ■

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 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 5.3, since in the general case $\neg(\alpha[A])$ is not an element formula, but rather 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, this construction may generate an exponential number of element formulas.

6 Equivalence Results

In this section we encode every operator of TL into a derived formula of SL. As a result, we obtain a compositional encoding from TL to SL that preserves the interpretation of formulas. In the following, we use the notation AnyE for the element formula $\emptyset^{\perp}[\top]$, that matches every element in $\mathcal{E}$.

The constant $\top$ of TL already has its equivalent in SL, and $\mathbf{0}$ can be readily encoded using a Presburger constraint that matches only the null vector.

The encoding of the *positive operators* is given in the following table. Location, $\alpha[A]$, may be encoded using the (singleton) support $(\alpha[A])$, while the encodings of the other operators basically rely on the derived Presburger operators given in Section 4.2.

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

$\mathbf{0}$	$=_{\text{def}}$	$\exists N.\, (N = 0) \,.\, \text{AnyE}$
$\alpha[A]$	$=_{\text{def}}$	$\exists N.\, (N = 1) \,.\, \alpha[A]$
$A \vee B$	$=_{\text{def}}$	$\exists \mathbf{N}.\, (\phi_A \vee \phi_B) \,.\, \mathbf{E}$
$A \mid B$	$=_{\text{def}}$	$\exists \mathbf{N}.\, (\phi_A + \phi_B) \,.\, \mathbf{E}$
A^*	$=_{\text{def}}$	$\exists \mathbf{N}.\, (\phi_A^*) \,.\, \mathbf{E}$

The encodings for $A \vee B$ and $A \mid B$ require A and B to be defined over the same support. This condition is not problematic. Indeed, given two support vectors $\mathbf{E} = (E_1, \ldots, E_p)$ and $\mathbf{F} = (F_1, \ldots, F_q)$, the support $(E_1, \ldots, E_p, F_1, \ldots, F_q)$ refines both $\mathbf{E}$ and $\mathbf{F}$. Thus, by Proposition 5.2, we can always assume that any two formulas are defined using a common support.

The following proposition states the soundness of the encodings for the positive operators. The proof relies on algebraic properties of the semi-ring $(2^{\mathcal{IT}}, \cup, \emptyset, |, \{\mathbf{0}\})$, such as distributivity of parallel composition over set union and the exponentiation rule, $S^{n_1} \mid S^{n_2} = S^{n_1+n_2}$.

Proposition 6.1 *Assume $A = \exists \mathbf{N}.\, \phi_A \,.\, \mathbf{E}$ and $B = \exists \mathbf{N}.\, \phi_B \,.\, \mathbf{E}$. The following equations hold:*

$$
\begin{aligned}
[\![\mathbf{0}]\!] &= \{\mathbf{0}\}, \\
[\![\alpha[A]]\!] &= \{a[d] \mid a \in \alpha \ ,\ d \in [\![A]\!]\}, \\
[\![A \vee B]\!] &= [\![A]\!] \cup [\![B]\!], \\
[\![A \mid B]\!] &= [\![A]\!] \mid [\![B]\!], \\
[\![A^*]\!] &= [\![A]\!]^*.
\end{aligned}
$$

Proof By definition, $[\![\mathbf{0}]\!]$ is the set $0 \,.\, [\![\mathrm{AnyE}]\!]$, that is the singleton $\{\mathbf{0}\}$.

By definition, $[\![\alpha[A]]\!]$ is the set $1 \,.\, [\![\alpha[A]]\!]$, that is the set $[\![\alpha[A]]\!]$, which is itself defined as $\{a[d] \mid a \in \alpha \ , \ d \in [\![A]\!]\}$ (if you feel a bit confused, recall that we have defined $\alpha[A]$ twice, once as an element formula and once as a counting formula).

The set $[\![A \vee B]\!]$ is the union of the sheaved compositions $\mathbf{n} \,.\, [\![\mathbf{E}]\!]$ for all the vectors $\mathbf{n}$ in $[\![\phi_A \vee \phi_B]\!]$, that is, in $[\![\phi_A]\!] \cup [\![\phi_B]\!]$. Therefore $[\![A \vee B]\!] = [\![A]\!] \cup [\![B]\!]$.

The set $[\![A \mid B]\!]$ is the union of the sheaved compositions $\mathbf{n} \,.\, [\![\mathbf{E}]\!]$ for all the vectors $\mathbf{n}$ in $[\![\phi_A + \phi_B]\!]$, that is, for all the vectors of the form $\mathbf{n}_A + \mathbf{n}_B$ with $\mathbf{n}_A \in [\![\phi_A]\!]$ and $\mathbf{n}_B \in [\![\phi_B]\!]$. The result follows from the fact that $(\mathbf{n}_A + \mathbf{n}_B) \,.\, [\![\mathbf{E}]\!] = \mathbf{n}_A \,.\, [\![\mathbf{E}]\!] \mid \mathbf{n}_B \,.\, [\![\mathbf{E}]\!]$.

The set $[\![A^*]\!]$ is the union of the sheaved compositions $\mathbf{n} \,.\, [\![\mathbf{E}]\!]$ for all the vectors $\mathbf{n}$ in $[\![\phi_A^*]\!]$, that is, for all vectors of the form $\mathbf{n}_1 + \cdots + \mathbf{n}_k$ with $k \in \mathbb{N}$ and $\mathbf{n}_i \in [\![\phi_A]\!]$ for all $i \in 1..k$. Hence, from our result on parallel composition, we obtain that $[\![A^*]\!]$ is the set of all trees of the form $d_1 \mid \cdots \mid d_k$ with $k \in \mathbb{N}$ and $d_i \in [\![A]\!]$ for all $i \in 1..k$. Therefore, $[\![A^*]\!] = \bigcup_{k \in \mathbb{N}} [\![A]\!]^k = [\![A]\!]^*$, as needed. ■

For the *negative operators*, $\neg$ and $\rhd$, we use the corresponding Presburger operators. A major difference with the previous cases is that we require formulas defined over a common bases (see Definition 5.1). This condition is not problematic since, by Proposition 5.3, it is always possible to define a common refining basis from two given bases.

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

$$\begin{array}{lll} \neg A & =_{\mathrm{def}} & \exists \mathbf{N}. \, (\neg \phi_A) \,.\, \mathbf{E} \\ A \rhd B & =_{\mathrm{def}} & \exists \mathbf{N}. \, (\phi_A \rhd \phi_B) \,.\, \mathbf{E} \end{array}$$

The following proposition states the soundness of the encodings of $\neg$ and $\rhd$, which relies on the fundamental property of bases.

Proposition 6.2 *Assume* $A = \exists \mathbf{N}. \, \phi_A \,.\, \mathbf{E}$ *and* $B = \exists \mathbf{N}. \, \phi_B \,.\, \mathbf{E}$, *where* $\mathbf{E}$ *is a basis :*

$$\begin{array}{rcl} [\![\neg A]\!] & = & \mathcal{IT} \setminus [\![A]\!], \\ [\![A \rhd B]\!] & = & \{d \mid \forall d' \in [\![A]\!] \,.\, (d \mid d') \in [\![B]\!]\} \ . \end{array}$$

Proof Since $\mathbf{E}$ is a basis, for each $d \in \mathcal{IT}$, there is a unique $\mathbf{n}_d$ such that $d \in \mathbf{n}_d \,.\, [\![\mathbf{E}]\!]$. It follows that $d \in [\![A]\!]$ if and only if $\mathbf{n}_d \in [\![\phi_A]\!]$. Thus, by definition of $\neg A$, we have $d \in [\![\neg A]\!]$ if and only if $\mathbf{n}_d \in [\![\neg \phi_A]\!]$, if and only if $\mathbf{n}_d \notin [\![\phi_A]\!]$ if and only if $d \notin [\![A]\!]$. For the second equation, $d \in [\![A \rhd B]\!]$ if and only if $\mathbf{n}_d \in [\![\phi_A \rhd \phi_B]\!]$, if and only if for all $\mathbf{n}_A \in [\![\phi_A]\!]$ the sum $\mathbf{n}_A + \mathbf{n}_d$ is in $[\![\phi_B]\!]$. So if $d \in [\![A \rhd B]\!]$, then for all $d' \in [\![A]\!]$, $\mathbf{n}_{d'} \in [\![\phi_A]\!]$ and thus $\mathbf{n}_{d|d'} = \mathbf{n}_d + \mathbf{n}_{d'} \in [\![\phi_B]\!]$ yielding $(d \mid d') \in [\![B]\!]$. Conversely, if for all $d' \in [\![A]\!]$, $(d \mid d') \in [\![B]\!]$, then let $\mathbf{n}_A \in [\![\phi_A]\!]$, and let $d' \in \mathbf{n}_A \,.\, [\![\mathbf{E}]\!]$. We have $d' \in [\![A]\!]$, so $(d \mid d') \in [\![B]\!]$, yielding $\mathbf{n}_{d|d'} \in [\![\phi_B]\!]$. But $\mathbf{n}_{d|d'} = \mathbf{n}_d + \mathbf{n}_{d'} = \mathbf{n}_d + \mathbf{n}_A$, so $d \in [\![A \rhd B]\!]$. ■

Using the encoding of negation, we give a new encoding of location, $\alpha[\_]$, that is defined over a basis.

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

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

We are now in position to state our main result.

Theorem 6.1 *For any formula A of TL, there is a formula B of SL, defined over a proper basis, such that $[\![A]\!] = [\![B]\!]$.*

Proof By induction on the syntax of A. The encodings for constants of TL all yield formulas of SL defined over proper bases. For the unary operators, we can, by induction, directly apply the appropriate encodings, which all yield formulas of SL defined over proper bases. For the binary operators, say $A_1 \rhd A_2$, by induction there are formulas B_1 and B_2 of SL, defined over proper bases $\mathbf{E}_1$ and $\mathbf{E}_2$, such that $[\![A_1]\!] = [\![B_1]\!]$ and $[\![A_2]\!] = [\![B_2]\!]$. By Proposition 5.3, there is a proper basis $\mathbf{E}$ that refines both $\mathbf{E}_1$ and $\mathbf{E}_2$. Thus, by Proposition 5.2, there are formulas C_1 and C_2 of SL, both defined over the basis $\mathbf{E}$, such that $[\![C_1]\!] = [\![A_1]\!]$ and $[\![C_2]\!] = [\![A_2]\!]$. We conclude by applying the appropriate encoding to C_1 and C_2, thus obtaining a formula of SL defined over the proper basis $\mathbf{E}$. ■

Theorem 6.2 *For any formula B of SL, there is a formula A of TL such that $[\![A]\!] = [\![B]\!]$.*

Proof By induction on the syntax of B. If B is the constant $\top$, we simply take $A = \top$. Assume B is the sheaved composition $\exists \mathbf{N}.\, \phi \,.\, \mathbf{E}$, where $\mathbf{E} = (\alpha_1[B_1], \ldots, \alpha_p[B_p])$. By induction, there exists p formulas of TL, say $A_1, \ldots, A_p$, such that $[\![A_i]\!] = [\![B_i]\!]$ for all $i \in 1..p$.

Let $\mathbf{F}$ be the sequence of location formulas $(\alpha_1[A_1], \ldots, \alpha_p[A_p])$ and let $\mathbf{n} \,.\, \mathbf{F}$ denotes the formula $\alpha_1[A_1]^{n_1} \mid \cdots \mid \alpha_p[A_p]^{n_p}$ of TL. Let $L(\mathbf{b}_1, P_1), \ldots, L(\mathbf{b}_q, P_q)$ be a semilinear set representing $[\![\phi]\!]$. For every linear set $L(\mathbf{b}_j, P_j)$ (with $i \in 1..q$) we define the formula $C_j = (\mathbf{b}_j \,.\, \mathbf{F}) \mid (\mathbf{p}_{j,1} \,.\, \mathbf{F})^* \mid \ldots \mid (\mathbf{p}_{j,l_j} \,.\, \mathbf{F})^*$, of TL, with denotation $L(\mathbf{b}_j, P_j) \,.\, [\![\mathbf{E}]\!]$. We obtain a formula equivalent to B by defining $A = \bigvee_{j \in 1..q} C_j$. ■

6.1 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 $a[d]$: $\neg a[\top]$.

$$a^{\perp}[\top] = \neg\mathbf{0} \wedge \neg(\neg\mathbf{0} \mid \neg\mathbf{0}) \wedge \neg(a[\top]) \tag{2}$$

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 :

$$\exists M_1, N_1, M_2, N_2.(M = M_1 + M_2) \wedge (N = N_1 + N_2) \wedge (M_1 + N_1 \neq 0) \wedge (M_2 + N_2 \neq 0)$$

which is equivalent to $M + N \geqslant 2$. By combining these three Presburger formulas, we obtain that the right-hand side of (2) 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.

$$a[\top]^* = \neg(\top \mid a^{\perp}[\top]) \tag{3}$$

We use the same basis than in the previous example and again concentrate on the counting constraints. The left-hand side of (3) 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 :

$$\exists M_1, N_1, M_2, N_2.(M = M_1 + M_2) \wedge (N = N_1 + N_2) \wedge (N_2 \geqslant 1) \ ,$$

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, 29] 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. Following 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.

7 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 [15] 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_{fin} is a set of final states included in Q, and R is a set of transition rules. Transition rules are two kinds:

(1) $\quad \alpha[q'] \to q$
(2) $\quad \phi(\#q_1, \ldots, \#q_p) \to q$

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 co-finite 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, ϕ 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, $\mathbf{0}$.

Example 7.1 *Let $\mathcal{A}$ be the automaton with states $Q = \{q_a, q_b, q_s\}$, set of final states $Q_{fin} = \{q_s\}$ 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 7.2, 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 $b[\,] \mid a[b[\,] \mid a[\,]]$.

7.1 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)
$$\frac{e_1 \to q_{j_1} \quad \ldots \quad e_n \to q_{j_n} \quad \phi \to q \in R \qquad (n \neq 1) \quad \#_Q(q_{j_1} \mid \ldots \mid q_{j_n}) \in [\![\phi]\!]}{e_1 \mid \ldots \mid e_n \to q}$$

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 $\mathcal{A}$.

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 7.2 *Let $\mathcal{A}$ be the automaton defined in Example 7.1. 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:*

$$\begin{array}{rcll} d & \to & a[\mathbf{0}] \mid b[a[q_s] \mid b[\mathbf{0}]] & \to \quad a[q_s] \mid b[a[q_s] \mid b[\mathbf{0}]] \\ & \to & a[q_s] \mid b[a[q_s] \mid b[q_s]] & \to \quad q_a \mid b[a[q_s] \mid b[q_s]] \\ & \to & q_a \mid b[a[q_s] \mid q_b] & \to \quad q_a \mid b[q_a \mid q_b] \\ & \stackrel{\dagger}{\to} & q_a \mid b[q_s] & \to \quad q_a \mid q_b \qquad \stackrel{\dagger}{\to} q_s \end{array}$$

In transitions 7 and 9 (marked with a $\dagger$*-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,* $\#_Q(q_a \mid q_b) = (1, 1, 0)$*).*

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* [14], 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 [30], 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.

7.2 Determinization

In this section, we show that given a Sheaves Automaton, $\mathcal{A}$, we can build a complete deterministic Sheaves Automaton, $\det(\mathcal{A})$, accepting the same language. The definition of $\det(\mathcal{A})$ is an adaptation of the classical subset construction (for finite state automata). Hence, the deterministic automaton may be exponentially bigger than the original one.

We say that a Sheaves 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 $[\![\phi]\!] \cap [\![\psi]\!] = \emptyset$.

We say that a Sheaves Automaton $\mathcal{A}$ with set of states $Q = \{q_1, \ldots, q_p\}$ is *complete* if and only if for every state $q \in Q$ and label $a \in \Lambda$, there is a type (1) rule $\alpha[q] \to q'$ with $a \in \alpha$ for some q', and for every $\mathbf{n} \in \mathbb{N}^p$, there is a type (2) rule $\phi \to q$ such that $\mathbf{n} \in [\![\phi]\!]$.

A property of deterministic automata is that for each tree d there is at most one state $q \in Q$ such that $d \to q$. A property of complete automata is that for every document d there is at least one state $q \in Q$ such that $d \to q$. As opposed to the situation with finite state automata, these properties do not provide a characterization of deterministic or complete automata (even if they are necessary conditions.)

Let $\mathcal{A}$ be the automaton $\langle Q, Q_{\text{fin}}, R\rangle$ with states $Q = \{q_1, \ldots, q_p\}$. Intuitively, the states of $\det(\mathcal{A})$ are subset of Q (ranged over by $\mathcal{Q}_1, \mathcal{Q}_2, ...$) and we have a transition $d \to^* \mathcal{Q}$ in $\det(\mathcal{A})$ if and only if $\mathcal{Q}$ is exactly the set of states reachable from d in $\mathcal{A}$. More formally, the automaton $\det(\mathcal{A})$ is defined as follows:

- the states of $\det(\mathcal{A})$ is the powerset 2^Q, that is, a state $\mathcal{Q}$ of $\det(\mathcal{A})$ is a subset of Q,
- the final states of $\det(\mathcal{A})$ are the state $\mathcal{Q}$ such that $\mathcal{Q} \cap Q_{\text{fin}} \neq \emptyset$, that is, a state $\mathcal{Q}$ of $\det(\mathcal{A})$ is final if and only if it contains a final state of $\mathcal{A}$,
- the set of rules of $\det(\mathcal{A})$ is defined as follows. In the following, we write $a[q_1] \to q_2$ if there is a transition $\beta[q_1] \to q_2 \in R$ such that $a \in \beta$. Likewise, we write $a[\mathcal{Q}_1] \to \mathcal{Q}_2$ if for all $q_2 \in \mathcal{Q}_2$ there is $q_1 \in \mathcal{Q}_1$ such that $a[q_1] \to q_2$ and if there is no $q_1 \in \mathcal{Q}_1, q_3 \notin \mathcal{Q}_2$ such that $a[q_1] \to q_3$.

  type (1) rules: For each pair of states $\mathcal{Q}_1, \mathcal{Q}_2$ in $\det(\mathcal{A})$, let $\alpha_{\mathcal{Q}_1,\mathcal{Q}_2}$ be the set of all labels, a, such that $a[\mathcal{Q}_1] \to \mathcal{Q}_2$. This set corresponds to the following label expression:

$$\alpha_{\mathcal{Q}_1,\mathcal{Q}_2} \;=\; \Big(\bigcap_{q_2\in\mathcal{Q}_2}\Big(\bigcup_{\substack{q_1\in\mathcal{Q}_1\\ \beta[q_1]\to q_2\in R}}\beta\Big)\Big) \cap \Big(\bigcap_{q_2\notin\mathcal{Q}_2}\Big(\bigcap_{\substack{q_1\in\mathcal{Q}_1\\ \beta[q_1]\to q_2\in R}}\beta^{\perp}\Big)\Big)\ .$$

  For all states $\mathcal{Q}_1, \mathcal{Q}_2$ of $\det(\mathcal{A})$ such that $\alpha_{\mathcal{Q}_1,\mathcal{Q}_2} \neq \emptyset$ the type (1) rule $\alpha_{\mathcal{Q}_1,\mathcal{Q}_2}[\mathcal{Q}_1] \to \mathcal{Q}_2$ is a rule of $\det(\mathcal{A})$. By construction, for every state $\mathcal{Q}_1$ and label a there is a unique type (1) rule $\alpha[\mathcal{Q}_1] \to \mathcal{Q}_2$ in $\det(\mathcal{A})$ such that $a \in \alpha$.

  type (2) rules: let $\phi_1 \to q_{j_1}, \ldots, \phi_n \to q_{j_n}$ be the type (2) rules of $\mathcal{A}$. (By definition, the free variables of ϕ_i are $\#q_1, \ldots, \#q_p$.) For every $i \in 1..n$ we define the Presburger constraint ψ_i, with (the $|2^Q|$) free variables $\#\emptyset, \ldots, \#Q$, as follows:

$$\psi_i =_{\text{def}}\ \exists(\#q)_{q\in Q}, (N_{q,\mathcal{Q}})_{\substack{\mathcal{Q}\in 2^Q\\ q\in\mathcal{Q}}}.\big(\phi_i(\#q_1,\ldots,\#q_n)\wedge \bigwedge_{q\in Q}\big(\#q = \sum_{\substack{\mathcal{Q}\in 2^Q\\ q\in\mathcal{Q}}} N_{q,\mathcal{Q}}\big) \wedge \bigwedge_{\mathcal{Q}\in 2^Q}\big(\#\mathcal{Q} = \textstyle\sum_{q\in\mathcal{Q}} N_{q,\mathcal{Q}}\big)\big)\ .$$

  For all subset $I \subseteq 1..n$, let $\psi_I = \big(\bigwedge_{i\in I}\psi_i\big) \wedge \big(\bigwedge_{i\notin I}\neg\psi_i\big)$ and $\mathcal{Q}_I = \{q_{j_i} \mid i \in I\}$. For every subset $I \subseteq 1..n$ such that $[\![\psi_I]\!] \neq \emptyset$, the type (2) rule $\psi_I \to \mathcal{Q}_I$ is a rule of $\det(\mathcal{A})$. By construction, for every vector $\mathbf{n} \in \mathbb{N}^{2^{|Q|}}$ there is a unique type (2) rule $\psi_I \to \mathcal{Q}_I$ such that $\mathbf{n} \in [\![\psi_I]\!]$. (Take for I is the set of all indices, i, such that $\mathbf{n} \in [\![\psi_i]\!]$.)

Proposition 7.1 *The automaton* $\det(\mathcal{A})$ *is deterministic and complete.*

Proof We start by proving that $\det(\mathcal{A})$ is deterministic. Assume $\alpha_{\mathcal{Q},\mathcal{Q}_1}[\mathcal{Q}] \to \mathcal{Q}_1$ and $\alpha_{\mathcal{Q},\mathcal{Q}_2}[\mathcal{Q}] \to \mathcal{Q}_2$ are two distinct type (1) rules of $\det(\mathcal{A})$. We prove that $\alpha_{\mathcal{Q},\mathcal{Q}_1} \cap \alpha_{\mathcal{Q},\mathcal{Q}_2} = \emptyset$.

Assume there exist a label a in $\alpha_{\mathcal{Q},\mathcal{Q}_1} \cap \alpha_{\mathcal{Q},\mathcal{Q}_2}$. Therefore $a[\mathcal{Q}] \to \mathcal{Q}_1$ and $a[\mathcal{Q}] \to \mathcal{Q}_2$, and we have $\mathcal{Q}_1 = \mathcal{Q}_2$, which contradicts the fact that the two type (1) rules are distinct. For type (2) rules, let ψ_I and ψ_J be Presburger constraints appearing in the left-hand sides of two distinct rules of $\det(\mathcal{A})$. We prove that $[\![\phi_I]\!] \cap [\![\phi_J]\!] = \emptyset$. Assume there exists $i \in I \setminus J$, by definition of ψ_I we have $[\![\psi_I]\!] \subseteq [\![\psi_i]\!]$ and $[\![\psi_J]\!] \subseteq [\![\neg\psi_i]\!]$, so $[\![\psi_I]\!] \cap [\![\psi_J]\!] = \emptyset$. Otherwise, there must be some indices $i \in J \setminus I$ (unless $I = J$), and we obtain the result using a similar reasoning.

Next, we prove that $\det(\mathcal{A})$ is a complete automaton. Assume $\mathcal{Q} \in 2^Q$ is a state of $\det(\mathcal{A})$ and a is a label. We prove that there is a type (1) rule $\alpha_{\mathcal{Q},\mathcal{Q}'}[\mathcal{Q}] \to \mathcal{Q}'$ in $\det(\mathcal{A})$ such that $a \in \alpha_{\mathcal{Q},\mathcal{Q}'}$. It is enough to choose for $\mathcal{Q}'$ the state corresponding to the set $\{q' \mid \exists q \in \mathcal{Q} \,.\, a \in \alpha \wedge \alpha[q] \to q' \in R\}$, such that $a[\mathcal{Q}] \to \mathcal{Q}'$. Finally, for type (2) rules, we prove that for every vector of $\mathbf{n} \in \mathbb{N}^{2^{|Q|}}$ there is a rule $\psi_I \to \mathcal{Q}_I$ such that $\mathbf{n} \in [\![\psi_I]\!]$. It is enough to choose for I the set $\{i \mid \mathbf{n} \in [\![\psi_i]\!]\}$. ■

Next, we prove a connection between the states in $\det(\mathcal{A})$ and the set of states reachable from a common term in $\mathcal{A}$.

Lemma 7.1 *For every tree $d \in \mathcal{IT}$, we have $d \to \mathcal{Q}$ in $\det(\mathcal{A})$ if and only if $\mathcal{Q}$ is the set $\{q \mid d \to_{\mathcal{A}} q\}$, of states reachable from d in $\mathcal{A}$.*

Proof By induction on the definition of d. Assume $d \to \mathcal{Q}$ in $\det(\mathcal{A})$ and $q \in \mathcal{Q}$, we prove that $d \to q$ in $\mathcal{A}$. The proof of the converse case is similar. We choose the same notation than in the introduction of Sect. 7.2.

Case $d = \mathbf{0}$: by construction there is a unique type (2) rule $\psi_I \to \mathcal{Q}$ in $\det(\mathcal{A})$ such that $\mathbf{0} \models \psi_I$, where I is a subset of $1..n$. By definition of ψ_I we have that $\mathbf{0} \models \psi_i$ iff $i \in I$, that is, I is the set of indices such that the following formula is true:

$$\exists (\#q)_{q\in Q}, (N_{q,\mathcal{Q}})_{\substack{\mathcal{Q}\in 2^Q \\ q\in\mathcal{Q}}} . \phi_i \wedge \bigwedge_{q\in Q} \left(\#q = \sum_{\substack{\mathcal{Q}\in 2^Q \\ q\in\mathcal{Q}}} N_{q,\mathcal{Q}}\right) \wedge \bigwedge_{\mathcal{Q}\in 2^Q} \left(0 = \sum_{q\in\mathcal{Q}} N_{q,\mathcal{Q}}\right)$$

This is only possible if I is the set of indices such that $\mathbf{0} \models \phi_i$. If $I = \emptyset$ then $\mathbf{0}$ is not reachable by $\mathcal{A}$ and we have $Q = \emptyset$, as needed. Otherwise, for all state $q_{j_i} \in \mathcal{Q}$ we have $\mathbf{0} \models \phi_{j_i}$ and $\phi_{j_i} \to q_{j_i}$ is a type (2) rule of $\mathcal{A}$, as needed.

Case $d = a[d']$: there must be a type (1) rule $\alpha[Q'] \to \mathcal{Q}$ in $\det(\mathcal{A})$, with $a \in \alpha$, and $d' \to \mathcal{Q}'$. By induction, $\mathcal{Q}' = \{q' \mid d' \to_{\mathcal{A}} q'\}$, and by definition of $\det(\mathcal{A})$, the label a must be in the label expression $\alpha_{\mathcal{Q}',\mathcal{Q}}$. By property of $\alpha_{\mathcal{Q}',\mathcal{Q}}$, since $q \in \mathcal{Q}$, we have $a[q'] \to q$ for all $q' \in \mathcal{Q}'$, and therefore $a[d'] \to q$ in $\mathcal{A}$, as needed.

Case $d = e_1 \mid \cdots \mid e_k$ with $k > 1$: let $\mathcal{Q}_{l_i} = \{q \mid e_i \to_{\mathcal{A}} q\}$, for all $i \in 1..k$. By induction hypothesis, we have $e_i \to \mathcal{Q}_{l_i}$ in $\det(\mathcal{A})$.

Let $\mathbf{m} = \#_{2^Q}(\mathcal{Q}_1 \mid \ldots \mid \mathcal{Q}_k)$. By construction there is a unique type (2) rule $\psi_I \to \mathcal{Q}_I$ such that $\mathbf{m} \in [\![\psi_I]\!]$ and $\mathcal{Q} = \mathcal{Q}_I$. Moreover, I is the set of all indices i such that

$\mathbf{m} \in [\![\psi_i]\!]$. Therefore $\mathcal{Q}$ is the set $\{q_{j_i} \mid \mathbf{m} \in [\![\psi_i]\!]\}$, that is, q is one of the type q_{j_k}, with $k \in 1..n$, that is the target of a type (2) rules $\phi_k \to q_{j_k} \in R$.

Assume $\mathbf{m} = (m_\emptyset, \ldots, m_Q)$. We have $\mathbf{m} \in [\![\psi_k]\!]$ if and only if there are integers n_q, for $q \in Q$, and $n_{q,\mathcal{Q}}$, for $(q, \mathcal{Q}) \in Q \times 2^Q$, such that $(n_{q_1}, \ldots, n_{q_p}) \in [\![\phi_k]\!]$, $n_q = \sum_{\mathcal{Q} \ni q} n_{q,\mathcal{Q}}$ for each $q \in Q$, and $m_\mathcal{Q} = \sum_{q \in \mathcal{Q}} n_{q,\mathcal{Q}}$ for each $\mathcal{Q} \in 2^Q$.

Therefore we can find a sequence $s = q_{l_1}, \ldots, q_{l_k}$ of elements in Q such that $e_i \to q_{l_i} \in \mathcal{Q}_{l_i}$ for every $i \in 1..k$ and $\#_Q(s) = (n_{q_1}, \ldots, n_{q_p}) \in [\![\phi_k]\!]$. Hence $q = q_{j_k}$ is reachable from d in $\mathcal{A}$, as needed.

■

Proposition 7.2 *For every tree $d \in \mathcal{IT}$, we have $d \in \mathcal{L}(\mathcal{A})$ if and only if $d \in \mathcal{L}(\det(\mathcal{A}))$.*

Proof Assume $d \in \mathcal{L}(\mathcal{A})$. Then $d \to_\mathcal{A} q \in Q_{\text{fin}}$ and, by Lemma 7.1, $d \to \mathcal{Q}$ in $\det(\mathcal{A})$ with $q \in \mathcal{Q}$. Therefore $\mathcal{Q}$ is a final state, as needed. Conversely, if $d \in \mathcal{L}(\det(\mathcal{A}))$, then $d \to \mathcal{Q}$ in $\det(()\mathcal{A})$ and there is a final state, q, in $\mathcal{A}$ such that $q \in \mathcal{Q}$. By Lemma 7.1, we have $d \to q$ in $\mathcal{A}$, as needed. ■

7.3 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 $Q^\times = Q \times Q' = \{(q_1, q'_1), \ldots, (q_p, q'_r)\}$ 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 ϕ and ϕ' 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:

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

The following property states the soundness of this construction.

Proposition 7.3 *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:

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

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 7.4 *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 7.5 *Given an automaton $\mathcal{A}$ we can build an automaton $\mathcal{A}^{\perp}$ such that $\mathcal{L}(\mathcal{A}^{\perp}) = \mathcal{IT} \setminus \mathcal{L}(\mathcal{A})$.*

Proof Assume $\mathcal{A} = \langle Q, Q_{\text{fin}}, R \rangle$ is a complete and deterministic automaton (otherwise we can apply the determinization procedure given in Sect. 7.2). The automaton $\mathcal{A}^{\perp} = \langle Q, Q \setminus Q_{\text{fin}}, R \rangle$ accepts exactly the trees that are not in $\mathcal{L}(\mathcal{A})$. ■

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. 6.1 and 7.1, 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.

7.4 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 ϕ, the evaluation of $\phi(n_1, \ldots, n_p)$ can be done in time $O(Cost(p,n))$ 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 ϕ. 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 7.6 *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 $O(|d|.|R|.Cost(|Q|,|d|))$ for a deterministic automaton. The problem is NP-complete for a non-deterministic automaton.*

Algorithm 1. Test for Emptiness

$Q_{\mathcal{E}} = \emptyset$
$Q_M = \{q \mid \phi \to q \in R \wedge \models \phi(\mathbf{0})\}$
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 $Q_{\mathcal{E}} := Q_{\mathcal{E}} \cup \{q\}$
  if $\phi \to q \in R$ and $\phi \setminus Q_{\mathcal{E}}$ 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*

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 $Q_{\mathcal{E}}$, where Q_M corresponds to reachable states and $Q_{\mathcal{E}}$ 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 ϕ. This amounts to checking the satisfiability of the Presburger formula $\phi(\#q_1, \ldots, \#q_p) \wedge \bigwedge_{q \notin Q_{\mathcal{E}}} \#q = 0$. When this formula is satisfiable, we say that the constraint $\phi \setminus Q_{\mathcal{E}}$ is satisfiable. In particular, the constraint $\phi \setminus \emptyset$ is satisfiable if and only if $\models \phi(\mathbf{0})$.

Proposition 7.7 *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 $Cost_{\mathcal{A}}$ denote the maximal time required to decide the satisfiability of the constraints occurring in type (2) rules of $\mathcal{A}$.

Proposition 7.8 *The problem $L(\mathcal{A}) \stackrel{?}{=} \emptyset$ is decidable in time $O(|Q|.|R|.Cost_{\mathcal{A}})$.*

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.

7.5 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 7.1 (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 $\emptyset^{\perp}[q] \to q$, and $(\#q \geqslant 0) \to q$.

The only other case is $A = \exists \mathbf{N}.\, \phi\,.\,\mathbf{E}$, where $\mathbf{E}$ is a support vector $(\alpha_1[A_1], \ldots, \alpha_p[A_p])$.

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 $\{a[d] \,|\, d \in [\![C]\!], a \in \alpha\}$: 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 $\alpha[q] \to q_s$ 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 $\{\mathcal{Q}_1, \ldots, \mathcal{Q}_m\}$ be the states of $\mathcal{A}$. A state $\mathcal{Q}$ of $\mathcal{A}$ is of the form $(q_1, \ldots, q_p)$, 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 $\mathcal{Q} \in \mathrm{fin}(i)$ to say that the i^{th} component of $\mathcal{Q}$ is a final state of $\mathcal{B}_i$. The constrained rules of $\mathcal{A}$ are of the form $\psi(M_1, \ldots, M_m) \to \mathcal{Q}$, 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 $\phi^{\exists}(M_1, \ldots, M_m) \to q_s$, where $\phi^{\exists}$ is satisfied by configurations $\mathcal{Q}_{j_1} \mid \cdots \mid \mathcal{Q}_{j_n}$ containing only states in $\mathrm{fin}(i)$ (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^i_j, for $j \in 1..p$, corresponding to final states of $\mathcal{B}_j$ occurring in $\mathcal{Q}_i$.

$$\exists \big(X^i_j\big)_{\substack{i\in 1..m\\ j\in 1..p}} \cdot \left(\begin{array}{l} \bigwedge_{i\in 1..m} \big(M_i = \sum_{\substack{j\in 1..p\\ \mathcal{Q}_i \in \mathrm{fin}(j)}} X^i_j\big) \\ \wedge\ \phi\big(\sum_{\substack{i\in 1..m\\ \mathcal{Q}_i \in \mathrm{fin}(1)}} X^i_1,\ \cdots\ , \sum_{\substack{i\in 1..m\\ \mathcal{Q}_i \in \mathrm{fin}(p)}} X^i_p\big) \end{array} \right)$$

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

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 $\top$ and $a\hat{}b$ stands for the exponentiation a^b. The decidability (and the complexity) of SL follows

from Th. 7.1 and Proposition 7.8. The complexity is doubly exponential in the size of A (like in Proposition 7.8, we abstract over the complexity of deciding the satisfiability of the Presburger constraints).

Theorem 7.2 (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 6). We obtain that the satisfiability problem for TL is in double exponential time. Once more, we abstract over the complexity of Presburger formulas.

Theorem 7.3 (Satisfiability) *For any formula A of TL, the satisfiability problem $[\![A]\!] = \emptyset$ is decidable in time $O(T^{2\hat{}}(3\hat{}|A|^2))$.*

8 Recursive Sheaves Logic

Constrained, type (2), rules of sheaves automata 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, that may contains 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

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

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 $X \leftarrow \exists N.\,(N \geqslant 0)\,.\,\emptyset^{\perp}[X]$ 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 ESL 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 8.1 (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 $\alpha[q_Y] \rightarrow q_{\alpha[Y]}$ 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: $\phi(\#q_{\alpha_1[Y_1]}, \ldots, \#q_{\alpha_p[Y_p]}) \rightarrow q_Y$. 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. ■

Using the definition of the derived operators given in Section 6, we can prove that ESL corresponds to an extension of TL with recursive definitions, where the location operator is limited to recursive variable, $a[X]$. For example, formula (4), below, is satisfied by trees with a path $(a.b)^*$ and (5) is satisfied by trees matching A somewhere.

$$\left\langle\, X \leftarrow (a[Y] \mid \top) \vee \mathbf{0}\ ,\ Y \leftarrow (b[X] \mid \top)\ ;\ X \,\right\rangle \tag{4}$$

$$\left\langle\, X \leftarrow (\emptyset^{\perp}[X] \mid \top) \vee A\ ;\ X \,\right\rangle \tag{5}$$

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 ESL in order to work with top-down sheaves automata. We take inspiration from a restriction on (sequential composition in) XML Schema [31, Sect. 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 (4) 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.

9 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 sub-structural 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 5, 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 8. 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.

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*, pages 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*, pages 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*, pages 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*, pages 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*, pages 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] G. Castagna and F. Nardelli. The seal calculus revisited: Contextual equivalence and bisimilarity. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 2556 of *LNCS*, pages 85–96. Springer, 2002.

[12] 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.

[13] 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*, pages 339–354. Springer, 2001.

[14] 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.

[15] 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.

[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*, pages 246–263. Springer, 2003. Appears also as Technical Report 4641, INRIA, Nov. 2002.

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

[19] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In *Conference on Concurrency Theory (CONCUR)*, volume 1119 of *LNCS*, pages 406–421. Springer, 1996.

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

[21] H. Hosoya and B. C. Pierce. Regular expression pattern matching for XML. In *Principles of Programming Languages (POPL)*, pages 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)*, pages 14–26. ACM Press, 2001.

[23] M. Kanovich. Horn programming in linear logic is NP-complete. In *Symposium on Logic in Computer Science (LICS)*, pages 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*, pages 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] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik. In *Comptes rendus du premier Congres de Mathématiques des Pays Slaves, Warsaw*, pages 92–101, 1929.

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

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

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

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

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399