Deciding Validity in a Spatial Logic for Trees

Cristiano Calcagno
Queen Mary, University of London

Luca Cardelli
Microsoft Research

Andrew D. Gordon
Microsoft Research

ABSTRACT

We consider a propositional spatial logic for finite trees. The logic includes $\mathcal{A} \mid \mathcal{B}$ (tree composition), $\mathcal{A} \triangleright \mathcal{B}$ (the implication induced by composition), and $\mathbf{0}$ (the unit of composition). We show that the satisfaction and validity problems are equivalent, and decidable. The crux of the argument is devising a finite enumeration of trees to consider when deciding whether a spatial implication is satisfied. We introduce a sequent calculus for the logic, and show it to be sound and complete with respect to an interpretation in terms of satisfaction. Finally, we describe a complete proof procedure for the sequent calculus. We envisage applications in the area of logic-based type systems for semistructured data. We describe a small programming language based on this idea.

Categories and Subject Descriptors

F.4 [**Theory of Computation**]: MATHEMATICAL LOGIC AND FORMAL LANGUAGES; F.3 [**Theory of Computation**]: LOGICS AND MEANINGS OF PROGRAMS

General Terms

Languages, Algorithms, Theory

1. INTRODUCTION

Due to the growing popularity of semistructured data [2], and particularly XML [1], there is a renewed interest in typed programming languages that can manipulate tree-like data structures. Unfortunately, semistructured data cannot be checked by conventional type systems with sufficient flexibility. More advanced type systems are being proposed that better match the data schemas used with semistructured data [16].

In general, we are going to have some tree-like data t, and some description language T that can flexibly describe the shape of the data. We are interested in description languages so flexible that they are akin to logics rather than to type systems. The question is: what is needed to use a description language T as a type system in some programming language that manipulates t data?

First of all, the programming language needs to analyze the data, so it needs to check at run-time whether a tree value matches a description. In type system terms this is a run-time typing problem: does tree t have type A. In logical terms this is a satisfaction problem: does tree t satisfy formula A.

Second, the programming language needs (most likely) to check at compile time whether a description A is less general than a description B. In terms of type system this is a subtyping test: is type A a subtype of type B. In logic terms this is a validity test: does every tree t satisfying formula A also satisfy formula B.

Given both a satisfaction and a validity algorithm, it is then fairly routine to build a type system around the description language, along with an operational semantics obeying standard typing soundness properties. The key problem, though, is to find rich description languages that admit satisfaction and (more crucially) validity algorithms. In the case of XDuce [17], for example, these algorithms are found in tree automata theory.

We propose here a logic that can be used as a rich description language for tree-like data. It emerges as an application of the novel area of "spatial" logics used for describing data and network structures. The logic of this paper is so expressive that, in fact, satisfaction and validity are equivalent problems (validity can be defined internally). For a restricted version of the spatial logics studied so far, we are able to obtain a validity algorithm, and this is sufficient for language applications. We end this paper by describing a simple language based on these ideas.

In a spatial logic, the truth of a formula depends on its location. Models for spatial logics include computational structures such as concurrent objects [5], heaps [22, 18, 21], trees [9], graphs [8], and also process calculi such as the π-calculus [3, 4] and the ambient calculus [10, 11]. Previous applications of spatial logics include specifying and verifying imperative and concurrent programs, and querying semistructured data.

The spatial logic of this paper describes properties of finite edge-labelled trees. In our textual notation, $n_1[P_1] \mid \cdots \mid n_k[P_k]$ is a tree consisting of k edges, labelled $n_1, \ldots, n_k$, leading to k subtrees $P_1, \ldots, P_k$, respectively. Our logic starts with propositional primitives: conjunction $\mathcal{A} \wedge \mathcal{B}$, implication $\mathcal{A} \Rightarrow \mathcal{B}$, and falsity $\mathbf{F}$. To this basis, we add spatial primitives: *composition* $\mathcal{A} \mid \mathcal{B}$ (satisfied by composite trees

TLDI'03, January 18, 2003, New Orleans, Louisiana, USA.

$P \mid Q$ where P and Q satisfy $\mathcal{A}$ and $\mathcal{B}$, respectively), *guarantee* $\mathcal{A} \triangleright \mathcal{B}$ (the spatial implication corresponding to composition, satisfied by trees that, whenever composed with any tree that satisfies $\mathcal{A}$, result in trees that satisfy $\mathcal{B}$) and *void* $\mathbf{0}$ (the unit of composition, satisfied by the empty tree). We complete the logic with primitives for labelled edges: *location* $n[\mathcal{A}]$ (satisfied by a tree $n[P]$ if P satisfies $\mathcal{A}$) and *placement* $\mathcal{A}@n$ (satisfied by a tree P if the tree $n[P]$ satisfies $\mathcal{A}$).

We consider the satisfaction problem (whether a given tree satisfies a given formula) and the validity problem (whether every tree satisfies a given formula). Since satisfaction of the guarantee operator $\mathcal{A} \triangleright \mathcal{B}$ is defined as an infinite quantification over all trees, neither problem is obviously decidable. Our first significant result, is that both are, in fact, decidable (Theorem 2). In effect, we show how to decide validity by model checking. The main auxiliary result (Theorem 1) is that we need consider only a finite enumeration of trees when model checking a formula $\mathcal{A} \triangleright \mathcal{B}$.

Subsequently, we introduce a sequent calculus for our spatial logic, and show how to decide validity by deduction in this calculus. The finite enumeration of trees introduced in the first half is built into the right rule for $\mathcal{A} \triangleright \mathcal{B}$. Our sequent calculus has a standard interpretation in terms of the satisfaction predicate. By appeal to Theorem 1, we show the sequent calculus to be sound (Theorem 3) and complete (Theorem 4) with respect to its interpretation. Moreover, we obtain and verify a complete algorithm for finding proofs in the sequent calculus (Theorem 5). The resulting algorithm for validity is better suited to optimisations than the algorithm based directly on model checking.

Section 2 gives formal definitions of our logic and its model. Section 3 develops our first algorithm for validity, based on model checking. Section 4 develops our second algorithm, based on our sequent calculus. Section 5 describes a small programming language for manipulating trees, to illustrate the idea of using spatial logic formulas as programming language types. Section 6 concludes.

A technical report [6] includes all the proofs omitted from this technical summary.

2. GROUND PROPOSITIONAL SPATIAL LOGIC (REVIEW)

This section introduces our spatial logic and its model. First, we define our notation for edge-labelled finite trees. Second, we introduce the formulas of the logic and their semantics: the satisfaction predicate, $P \models \mathcal{A}$, means that the tree P satisfies the formula $\mathcal{A}$. Third, we define the validity predicate, $\mathbf{vld}(\mathcal{A})$, to mean $P \models \mathcal{A}$ for every tree P. By constructing certain characteristic formulas, we note that satisfaction and validity are interderivable.

In a study of a richer spatial logic than the one considered here, Hirschkoff, Lozes, and Sangiorgi [15] also define characteristic formulas for ambient processes, and note equivalences between the satisfaction and validity problems.

2.1 Edge-Labelled Finite Trees

Let m, n range over an infinite set N of names. The model of our logic is the set of edge-labelled trees, finitely branching and of finite depth.

TREES:

$P, Q ::=$	tree
$\mathbf{0}$	empty tree
$P \mid Q$	composition
$m[P]$	edge labelled by m, atop tree P

Let $fn(P)$ be the set of names free in P. For any $X \subseteq \mathsf{N}$, let $\mathsf{Tree}_X \triangleq \{P \mid fn(P) \subseteq X\}$.

STRUCTURAL EQUIVALENCE: $P \equiv Q$

$P \equiv P$	(Struct Refl)
$Q \equiv P \Rightarrow P \equiv Q$	(Struct Symm)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(Struct Trans)
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(Struct Par)
$P \equiv Q \Rightarrow M[P] \equiv M[Q]$	(Struct Amb)
$P \mid Q \equiv Q \mid P$	(Struct Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct Par Assoc)
$P \mid \mathbf{0} \equiv P$	(Struct Zero Par)

LEMMA 1. *If* $P \in \mathsf{Tree}_X$ *and* $P \equiv Q$ *then* $Q \in \mathsf{Tree}_X$.

2.2 Logical Formulas and Satisfaction

LOGICAL FORMULAS:

$\mathcal{A}, \mathcal{B} ::=$	formula
$\mathbf{F}$	false
$\mathcal{A} \wedge \mathcal{B}$	conjunction
$\mathcal{A} \Rightarrow \mathcal{B}$	implication
$\mathbf{0}$	void
$\mathcal{A} \mid \mathcal{B}$	composition
$\mathcal{A} \triangleright \mathcal{B}$	guarantee
$n[\mathcal{A}]$	location
$\mathcal{A}@n$	placement

The derived propositional connectives $\mathbf{T}$, $\neg\mathcal{A}$, $\mathcal{A} \vee \mathcal{B}$, are defined in the usual way. Name equality can be defined by $m = n \triangleq m[\mathbf{T}]@n$; this formula holds if and only if $m = n$. We write $\mathcal{A}\{m \leftarrow m'\}$ for the outcome of substituting each occurrence of the name m in the formula $\mathcal{A}$ with the name m'.

We define the satisfaction predicate, $P \models \mathcal{A}$, as follows.

SATISFACTION: $P \models \mathcal{A}$

$P \models \mathbf{F}$	never	
$P \models \mathcal{A} \wedge \mathcal{B}$	$\triangleq$	$P \models \mathcal{A} \wedge P \models \mathcal{B}$
$P \models \mathcal{A} \Rightarrow \mathcal{B}$	$\triangleq$	$P \models \mathcal{A} \Rightarrow P \models \mathcal{B}$
$P \models \mathbf{0}$	$\triangleq$	$P \equiv \mathbf{0}$
$P \models \mathcal{A} \mid \mathcal{B}$	$\triangleq$	$\exists P', P''.P \equiv P' \mid P'' \wedge P' \models \mathcal{A} \wedge P'' \models \mathcal{B}$
$P \models \mathcal{A} \triangleright \mathcal{B}$	$\triangleq$	$\forall P'.P' \models \mathcal{A} \Rightarrow P \mid P' \models \mathcal{B}$
$P \models n[\mathcal{A}]$	$\triangleq$	$\exists P'.P \equiv n[P'] \wedge P' \models \mathcal{A}$
$P \models \mathcal{A}@n$	$\triangleq$	$n[P] \models \mathcal{A}$

A basic property is that structural congruence preserves satisfaction:

LEMMA 2. *If* $P \models \mathcal{A}$ *and* $P \equiv P'$ *then* $P' \models \mathcal{A}$.

It is useful to know that every tree P has a characteristic formula $\underline{P}$. Let $\underline{\mathbf{0}} \triangleq \mathbf{0}$, $\underline{P \mid Q} \triangleq \underline{P} \mid \underline{Q}$, and $\underline{m[P]} \triangleq m[\underline{P}]$. The formula $\underline{P}$ identifies P up to structural equivalence:

LEMMA 3. *For all P and Q, $Q \models \underline{P}$ if and only if $Q \equiv P$.*

Now, to turn the definition of satisfaction into an algorithm, that is, to build a model checker for the logic, we must show that the three quantifications in the clauses for $\mathcal{A} \mid \mathcal{B}$, $\mathcal{A} \triangleright \mathcal{B}$, and $n[\mathcal{A}]$ can be reduced to finite problems. It is not hard to reduce the clauses for $\mathcal{A} \mid \mathcal{B}$ and $n[\mathcal{A}]$ to finite quantifications [10], but it seems far from obvious how to reduce satisfaction of $\mathcal{A} \triangleright \mathcal{B}$ to a finite problem. The principal result of the paper, Theorem 1, is that for any $\mathcal{A}'$, $\mathcal{A}''$ there is a finite set $T(\mathcal{A}' \triangleright \mathcal{A}'')$ such that:

$$P \models \mathcal{A}' \triangleright \mathcal{A}'' \Leftrightarrow \forall P' \in T(\mathcal{A}' \triangleright \mathcal{A}'').P' \models \mathcal{A}' \Rightarrow P' \mid P \models \mathcal{A}''$$

2.3 Validity of a Formula

The validity predicate, $\mathbf{vld}(\mathcal{A})$, means every tree satisfies the formula $\mathcal{A}$.

VALIDITY: $\mathbf{vld}(\mathcal{A})$

$$\mathbf{vld}(\mathcal{A}) \triangleq \forall P.P \models \mathcal{A}$$

The next two lemmas exhibit formulas to encode validity in terms of satisfaction, and the converse.

LEMMA 4 (VALIDITY FROM SATISFACTION). $\mathbf{vld}(\mathcal{A})$ *if and only if* $\mathbf{0} \models \mathbf{T} \triangleright \mathcal{A}$.

LEMMA 5 (SATISFACTION FROM VALIDITY). $P \models \mathcal{A}$ *if and only if* $\mathbf{vld}(\underline{P} \Rightarrow \mathcal{A})$.

Hence, the validity and satisfaction problems are equivalent. The goal of the paper is to show both are decidable.

3. DECIDING VALIDITY BY MODEL CHECKING

The crux of our problem is the infinite quantification in the definition of satisfaction for $\mathcal{A} \triangleright \mathcal{B}$. We bound this infinite quantification in three steps, which lead to an alternative definition in terms of a finite quantification. This leads to a model checking procedure, and hence to an algorithm for validity.

- In Section 3.1, we bound the alphabet of distinct names that may occur in trees that need to be considered. Let $fn(\mathcal{A})$ be the set of names occurring free in any formula $\mathcal{A}$. Let m be some other name. Proposition 1 asserts that $P \models \mathcal{A} \triangleright \mathcal{B}$ if and only if $Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B}$ for all trees Q with edge-labels drawn from the set $fn(\mathcal{A}) \cup \{m\}$.
- In Section 3.2, we introduce a measure of the size of a tree, and bound both the alphabet and size of trees that need to be considered. Proposition 4 asserts that $P \models \mathcal{A} \triangleright \mathcal{B}$ if and only if $Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B}$ for all the trees Q smaller than a size determined by $\mathcal{A}$ and with edge-labels drawn from a particular finite alphabet.
- In Section 3.3, we give a procedure to enumerate a finite set of structural equivalence classes of trees determined by a formula. Theorem 1 asserts that $P \models \mathcal{A} \triangleright \mathcal{B}$ if and only if $Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B}$ for all the representatives Q of these equivalence classes. Hence, we prove in Theorem 2 that satisfaction, and hence validity, is decidable.

3.1 Bounding the Names to Consider

We need the following facts relating substitution with the operators for adding an edge to a tree and for composing trees.

LEMMA 6. *If $n \notin \{m, m'\}$ then:*

$$P\{m \leftarrow m'\} \equiv n[Q] \Leftrightarrow \exists P'.P \equiv n[P'] \wedge P'\{m \leftarrow m'\} \equiv Q$$

LEMMA 7.

$$\begin{aligned} &P\{m \leftarrow m'\} \equiv Q' \mid Q'' \Leftrightarrow \\ &\quad \exists P', P''.\, P \equiv P' \mid P'' \wedge P'\{m \leftarrow m'\} \equiv Q' \\ &\qquad \wedge P''\{m \leftarrow m'\} \equiv Q' \end{aligned}$$

Given these facts we can show that satisfaction of a formula is independent of any name not occurring in the formula.

LEMMA 8. *If $m, m' \notin fn(\mathcal{A})$, $P \models \mathcal{A} \Leftrightarrow P\{m \leftarrow m'\} \models \mathcal{A}$.*

This lemma is not true for the logic extended with quantifiers: we have $m[] \mid n[] \models \exists x, y.(x[] \mid y[]) \wedge x \neq y$ but $m[] \mid m[] \not\models \exists x, y.(x[] \mid y[]) \wedge x \neq y$.

PROPOSITION 1 (BOUNDING NAMES). *If $m \notin fn(\mathcal{A} \triangleright \mathcal{B})$ then:*

$$P \models \mathcal{A} \triangleright \mathcal{B} \Leftrightarrow (\forall Q \in \mathsf{Tree}_{fn(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}}.\ Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B})$$

PROOF. The forwards direction is immediate. For the backwards direction, assume that $(\forall Q \in \mathsf{Tree}_{fn(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}}.\ Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B})$ and consider any tree Q such that $Q \models \mathcal{A}$. Suppose that $fn(P|Q) \subseteq fn(\mathcal{A} \triangleright \mathcal{B}) \cup \{m, n_1, \ldots, n_k\}$ where $\{n_1, \ldots, n_k\} \cap (fn(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}) = \emptyset$. Let $P' = P\{n_1 \leftarrow m\} \cdots \{n_k \leftarrow m\}$ and $Q' = Q\{n_1 \leftarrow m\} \cdots \{n_k \leftarrow m\}$. By repeated application of Lemma 8, we get that $Q \models \mathcal{A} \Leftrightarrow Q' \models \mathcal{A}$. Since $Q' \in \mathsf{Tree}_{fn(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}}$ and $Q' \models \mathcal{A}$, we obtain $P \mid Q' \models \mathcal{B}$ by assumption. Now, we have:

$$\begin{aligned} &(P \mid Q')\{n_1 \leftarrow m\} \cdots \{n_k \leftarrow m\} \\ &\quad = \quad P' \mid Q' \\ &\quad = \quad (P \mid Q)\{n_1 \leftarrow m\} \cdots \{n_k \leftarrow m\} \end{aligned}$$

Hence, by repeated application of Lemma 8, we get that $P \mid Q' \models \mathcal{B} \Leftrightarrow P' \mid Q' \models \mathcal{B} \Leftrightarrow P \mid Q \models \mathcal{B}$. Hence $P \mid Q \models \mathcal{B}$ follows. □

3.2 Bounding the Sizes to Consider

We introduce measures of the height and width of both trees and formulas.

DEFINITION 1 (NOTATION). *Write $a \cdot P$ for $a \geq 0$ copies of P in parallel: $P \mid \ldots \mid P$.*

DEFINITION 2 (SIZE OF TREES). $|P|^{\mathsf{hw}} \triangleq (h, w)$ *iff there are a_1, n_1, P_1, …, a_k, n_k, P_k, for some k, such that:*

- $P \equiv a_1 \cdot n_1[P_1] \mid \ldots \mid a_k \cdot n_k[P_k]$
- $\forall i, j \in 1..k.\ n_i[P_i] \equiv n_j[P_j] \Rightarrow i = j$
- $(h_i, w_i) = |P_i|^{\mathsf{hw}}$ *for each* $i \in 1..k$
- *if* $k = 0$, $h = 0$; *otherwise* $h = 1 + max(h_1, \ldots, h_k)$
- *if* $k = 0$, $w = 0$; *otherwise* $w = max(a_1, \ldots, a_k, w_1, \ldots, w_k)$

When $|P|^{\mathsf{hw}} = (h, w)$*, we write* $|P|^{\mathsf{h}}$ *for* h *and* $|P|^{\mathsf{w}}$ *for* w*. We write* $(h_1, w_1) \leq (h_2, w_2)$ *for* $(h_1 \leq h_2) \wedge (w_1 \leq w_2)$*.*

Intuitively $|P|^{\mathsf{h}}$ is the height of P, and $|P|^{\mathsf{w}}$ is the width, defined as the maximum *multiplicity* of the subtrees of P. The multiplicity is the number of structurally equivalent nonempty trees under the same edge. For example:

- $|\mathbf{0}|^{\mathsf{hw}} = (0, 0)$
- $|n[\mathbf{0}]|^{\mathsf{hw}} = (1, 1)$
- $|n[\mathbf{0}] \mid m[\mathbf{0}]|^{\mathsf{hw}} = (1, 1)$
- $|n[\mathbf{0}] \mid n[\mathbf{0}]|^{\mathsf{hw}} = (1, 2)$
- $|n[m[\mathbf{0}]]|^{\mathsf{hw}} = (2, 1)$
- $|n[n[\mathbf{0}]]|^{\mathsf{hw}} = (2, 1)$

Next, we define height and width measures for logical formulas.

Height of Logical Formulas

$$
\begin{array}{lcl}
|\mathbf{F}|^{\mathsf{h}} & \triangleq & 0 \\
|\mathcal{A} \wedge \mathcal{B}|^{\mathsf{h}} & \triangleq & max(|\mathcal{A}|^{\mathsf{h}}, |\mathcal{B}|^{\mathsf{h}}) \\
|\mathcal{A} \Rightarrow \mathcal{B}|^{\mathsf{h}} & \triangleq & max(|\mathcal{A}|^{\mathsf{h}}, |\mathcal{B}|^{\mathsf{h}}) \\
|\mathbf{0}|^{\mathsf{h}} & \triangleq & 1 \\
|\mathcal{A} \mid \mathcal{B}|^{\mathsf{h}} & \triangleq & max(|\mathcal{A}|^{\mathsf{h}}, |\mathcal{B}|^{\mathsf{h}}) \\
|\mathcal{A} \triangleright \mathcal{B}|^{\mathsf{h}} & \triangleq & |\mathcal{B}|^{\mathsf{h}} \\
|n[\mathcal{A}]|^{\mathsf{h}} & \triangleq & 1 + |\mathcal{A}|^{\mathsf{h}} \\
|\mathcal{A}@n|^{\mathsf{h}} & \triangleq & max(|\mathcal{A}|^{\mathsf{h}} - 1, 0)
\end{array}
$$

Width of Logical Formulas

$$
\begin{array}{lcl}
|\mathbf{F}|^{\mathsf{w}} & \triangleq & 0 \\
|\mathcal{A} \wedge \mathcal{B}|^{\mathsf{w}} & \triangleq & max(|\mathcal{A}|^{\mathsf{w}}, |\mathcal{B}|^{\mathsf{w}}) \\
|\mathcal{A} \Rightarrow \mathcal{B}|^{\mathsf{w}} & \triangleq & max(|\mathcal{A}|^{\mathsf{w}}, |\mathcal{B}|^{\mathsf{w}}) \\
|\mathbf{0}|^{\mathsf{w}} & \triangleq & 1 \\
|\mathcal{A} \mid \mathcal{B}|^{\mathsf{w}} & \triangleq & |\mathcal{A}|^{\mathsf{w}} + |\mathcal{B}|^{\mathsf{w}} \\
|\mathcal{A} \triangleright \mathcal{B}|^{\mathsf{w}} & \triangleq & |\mathcal{B}|^{\mathsf{w}} \\
|n[\mathcal{A}]|^{\mathsf{w}} & \triangleq & max(2, |\mathcal{A}|^{\mathsf{w}}) \\
|\mathcal{A}@n|^{\mathsf{w}} & \triangleq & |\mathcal{A}|^{\mathsf{w}}
\end{array}
$$

Here are the sizes for the derived propositional connectives:

$$
\begin{array}{lcl}
|\mathbf{T}|^{\mathsf{h}} & \triangleq & 0 \\
|\neg\mathcal{A}|^{\mathsf{h}} & \triangleq & |\mathcal{A}|^{\mathsf{h}} \\
|\mathcal{A} \vee \mathcal{B}|^{\mathsf{h}} & \triangleq & max(|\mathcal{A}|^{\mathsf{h}}, |\mathcal{B}|^{\mathsf{h}}) \\
|\mathbf{T}|^{\mathsf{w}} & \triangleq & 0 \\
|\neg\mathcal{A}|^{\mathsf{w}} & \triangleq & |\mathcal{A}|^{\mathsf{w}} \\
|\mathcal{A} \vee \mathcal{B}|^{\mathsf{w}} & \triangleq & max(|\mathcal{A}|^{\mathsf{w}}, |\mathcal{B}|^{\mathsf{w}})
\end{array}
$$

We define a relation $\sim_{h,w}$ between trees, parameterized by the size (h, w). The main property of the relation is that if $P \sim_{h,w} Q$ then no formula with size (h, w) can distinguish between P and Q (Proposition 2).

Definition 3 (Relation $P \sim_{h,w} Q$).

$$
\begin{array}{lcl}
P \sim_{0,w} Q & & \textit{always} \\
P \sim_{h+1,w} Q & \Leftrightarrow & \forall i \in 1..w.\, \forall n, P_j \textit{ with } j \in 1..i. \\
 & & \quad \textit{if } P \equiv n[P_1] \mid \cdots \mid n[P_i] \mid P_{i+1} \\
 & & \quad \textit{then } Q \equiv n[Q_1] \mid \cdots \mid n[Q_i] \mid Q_{i+1} \\
 & & \quad \textit{such that } P_j \sim_{h,w} Q_j \textit{ for } j \in 1..i \\
 & & \textit{and vice versa}
\end{array}
$$

Note that $\sim_{h,w}$ is an equivalence relation: reflexivity, symmetry, and transitivity are immediate consequences of the definition. Moreover, it is preserved by structural congruence:

Lemma 9. *If* $P \sim_{h,w} Q$ *and* $Q \equiv R$ *then* $P \sim_{h,w} R$*.*

The following lemma shows that the relation $\sim_{h,w}$ is monotone in (h, w).

Lemma 10 (Monotonicity). *Suppose* $P \sim_{h,w} Q$ *and* $(h', w') \leq (h, w)$*. Then* $P \sim_{h',w'} Q$*.*

The following lemma shows that the relation $\sim_{h,w}$ is a congruence.

Lemma 11 (Congruence). *The following hold:*

(1) If $P \sim_{h,w} Q$ *then* $n[P] \sim_{h+1,w} n[Q]$*.*

(2) If $P \sim_{h,w} P'$ *and* $Q \sim_{h,w} Q'$ *then* $P \mid Q \sim_{h,w} P' \mid Q'$*.*

Lemma 12 (Inversion). *If* $P' \mid P'' \sim_{h,w_1+w_2} Q$ *then there exist* Q', Q'' *such that* $Q \equiv Q' \mid Q''$ *and* $P' \sim_{h,w_1} Q'$ *and* $P'' \sim_{h,w_2} Q''$*.*

Proposition 2. *Suppose* $|\mathcal{A}|^{\mathsf{hw}} = (h, w)$ *and* $P \models \mathcal{A}$ *and* $P \sim_{h,w} Q$*. Then* $Q \models \mathcal{A}$*.*

Proof. By induction on the structure of $\mathcal{A}$. □

The following lemma shows that each equivalence class determined by $\sim_{h,w}$ contains a tree of size bounded by (h, w).

Lemma 13 (Pruning). *For all* $P \in \mathsf{Tree}_X, h, w$ *there exists* $P' \in \mathsf{Tree}_X$ *such that* $P \sim_{h,w} P'$ *and* $|P'|^{\mathsf{hw}} \leq (h, w)$*.*

Proposition 3 (Bounding Size). *For any tree* P*, set of names* X *and formulas* $\mathcal{A}$ *and* $\mathcal{B}$*, if* $h = max(|\mathcal{A}|^{\mathsf{h}}, |\mathcal{B}|^{\mathsf{h}})$ *and* $w = max(|\mathcal{A}|^{\mathsf{w}}, |\mathcal{B}|^{\mathsf{w}})$ *then*

$$
\begin{array}{l}
(\forall Q \in \mathsf{Tree}_X.\, Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B}) \quad \Leftrightarrow \\
(\forall Q \in \mathsf{Tree}_X.\, |Q|^{\mathsf{hw}} \leq (h, w) \wedge Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B})
\end{array}
$$

Proof. The forwards direction is immediate. For the backwards direction, assume that the right hand side holds. Take any $Q \in \mathsf{Tree}_X$ such that $Q \models \mathcal{A}$. Then we have:

$\exists Q'.\, Q \sim_{h,w} Q' \wedge |Q'|^{\mathsf{hw}} \leq (h, w)$ by Lemma 13
$Q \sim_{|\mathcal{A}|^{\mathsf{h}}, |\mathcal{A}|^{\mathsf{w}}} Q'$ by Lemma 10 since $|\mathcal{A}|^{\mathsf{hw}} \leq (h, w)$
$Q' \models \mathcal{A}$ by Proposition 2
$P \mid Q' \models \mathcal{B}$ by assumption
$P \mid Q \sim_{h,w} P \mid Q'$ by Lemma 11
$P \mid Q \sim_{|\mathcal{B}|^{\mathsf{h}}, |\mathcal{B}|^{\mathsf{w}}} P \mid Q'$ by Lemma 10 since $|\mathcal{B}|^{\mathsf{hw}} \leq (h, w)$
$P \mid Q \models \mathcal{B}$ by Proposition 2

□

PROPOSITION 4 (BOUNDING SIZE AND NAMES). *For any tree P and formulas $\mathcal{A}$ and $\mathcal{B}$, suppose $m \notin fn(\mathcal{A} \triangleright \mathcal{B})$ and $X = fn(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}$ and $h = max(|\mathcal{A}|^{\mathsf{h}}, |\mathcal{B}|^{\mathsf{h}})$ and $w = max(|\mathcal{A}|^{\mathsf{w}}, |\mathcal{B}|^{\mathsf{w}})$. Then:*

$$P \models \mathcal{A} \triangleright \mathcal{B} \Leftrightarrow (\forall Q \in \mathsf{Tree}_X . |Q|^{\mathsf{hw}} \leq (h, w) \wedge Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B})$$

PROOF. We have:

$$\begin{aligned} & P \models \mathcal{A} \triangleright \mathcal{B} \\ \Leftrightarrow \quad & (\forall Q \in \mathsf{Tree}_X . Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B}) \\ \Leftrightarrow \quad & (\forall Q \in \mathsf{Tree}_X . |Q|^{\mathsf{hw}} \leq (h, w) \wedge Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B}) \end{aligned}$$

Proposition 1 justifies the first step, Proposition 3 the second. □

So, to check satisfaction of $\mathcal{A} \triangleright \mathcal{B}$, we need only consider trees whose free names are drawn from $fn(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}$, and whose size is bounded by $max(|\mathcal{A}|^{\mathsf{hw}}, |\mathcal{A}|^{\mathsf{hw}})$. We show in the next section, that the number of such trees, modulo structural equivalence, is finite. Hence, we obtain an algorithm for satisfaction of $\mathcal{A} \triangleright \mathcal{B}$.

3.3 Enumerating Equivalence Classes

In this section we present an explicit characterization of the equivalence classes on trees, modulo structural equivalence, determined by $\sim_{h,w}$.

DEFINITION 4 (NOTATION). *Consider the following notation, where metavariable c ranges over sets of trees modulo structural congruence:*

$$\begin{array}{lcl} \langle P \rangle_{\equiv} & \triangleq & \{P' \mid P \equiv P'\} \\ \langle P \rangle_{h,w} & \triangleq & \{P' \mid P \sim_{h,w} P'\} \\ c_1 + c_2 & \triangleq & c_1 \cup c_2 \\ n[c] & \triangleq & \{\langle n[P] \rangle_{\equiv} \mid \langle P \rangle_{\equiv} \in c\} \\ c^{\leq n} & \triangleq & \{\langle a_1 \cdot P_1 \mid \cdots \mid a_k \cdot P_k \rangle_{\equiv} \\ & & \quad \mid 0 \leq a_i \leq n \text{ for } i \in 1..k\} \\ & & \quad \text{when } c = \{\langle P_1 \rangle_{\equiv}, \ldots, \langle P_k \rangle_{\equiv}\} \end{array}$$

We can now give a direct definition of the set of equivalence classes $EQ^X_{h,w}$ determined by $\sim_{h,w}$, given a set of names X.

DEFINITION 5. *If $X = \{n_1, \ldots, n_k\}$, define $EQ^X_{h,w}$ as follows:*

$$\begin{array}{lcl} EQ^X_{0,w} & \triangleq & \{\langle 0 \rangle_{\equiv}\} \\ EQ^X_{h+1,w} & \triangleq & (n_1[EQ^X_{h,w}] + \cdots + n_k[EQ^X_{h,w}])^{\leq w} \end{array}$$

LEMMA 14. *If $|P|^{\mathsf{hw}} \leq (h, w)$ and $|P'|^{\mathsf{hw}} \leq (h, w)$, then*

(1) $P \in \mathsf{Tree}_X$ implies $\langle P \rangle_{\equiv} \in EQ^X_{h,w}$.

(2) $P \equiv P' \iff P \sim_{h,w} P'$.

The following lemma shows that $EQ^X_{h,w}$ contains exactly the trees (modulo $\equiv$) of size at most (h, w) with free names drawn from X.

LEMMA 15. $\langle P \rangle_{\equiv} \in EQ^X_{h,w} \Leftrightarrow P \in \mathsf{Tree}_X \wedge |P|^{\mathsf{hw}} \leq (h, w)$.

THEOREM 1 (FINITE BOUND). *Consider any formulas $\mathcal{A}$ and $\mathcal{B}$. Let $EQ^X_{h,w} = \{\langle Q_1 \rangle_{\equiv}, \ldots, \langle Q_n \rangle_{\equiv}\}$, where $h = max(|\mathcal{A}|^{\mathsf{h}}, |\mathcal{B}|^{\mathsf{h}})$ and $w = max(|\mathcal{A}|^{\mathsf{w}}, |\mathcal{B}|^{\mathsf{w}})$ and $X = fn(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}$ for some $m \notin fn(\mathcal{A} \triangleright \mathcal{B})$.*
Then, for any tree P:

$$P \models \mathcal{A} \triangleright \mathcal{B} \Leftrightarrow (\forall i \in 1..n . Q_i \models \mathcal{A} \Rightarrow P \mid Q_i \models \mathcal{B})$$

PROOF. Using Proposition 4, Lemma 15, and Lemma 2:

$$\begin{aligned} & P \models \mathcal{A} \triangleright \mathcal{B} \\ \Leftrightarrow \quad & (\forall Q \in \mathsf{Tree}_X . |Q|^{\mathsf{hw}} \leq (h, w) \wedge Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B}) \\ \Leftrightarrow \quad & (\forall Q . \langle Q \rangle_{\equiv} \in EQ^X_{h,w} \wedge Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B}) \\ \Leftrightarrow \quad & (\forall Q . (\exists i \in 1..n . Q \equiv Q_i) \wedge Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B}) \\ \Leftrightarrow \quad & (\forall i \in 1..n . \forall Q . Q \equiv Q_i \wedge Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B}) \\ \Leftrightarrow \quad & (\forall i \in 1..n . Q_i \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B}) \end{aligned}$$

□

Given this result, we can now show that each of the three quantifications in the definition of satisfaction can be reduced to a finite problem.

FINITE TEST SETS: $T(P)$, $T(\mathcal{A} \triangleright \mathcal{B})$, AND $T(n, P)$

$T(P)$ is the finite set $\{\langle Q, R \rangle \mid P \equiv Q \mid R\}/(\equiv \times \equiv)$.
$T(\mathcal{A} \triangleright \mathcal{B})$ is the finite set $EQ^X_{h,w}$,
where $h = max(|\mathcal{A}|^{\mathsf{h}}, |\mathcal{B}|^{\mathsf{h}})$and $w = max(|\mathcal{A}|^{\mathsf{w}}, |\mathcal{B}|^{\mathsf{w}})$
and $X = fn(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}$ for some $m \notin fn(\mathcal{A} \triangleright \mathcal{B})$.
$T(n, P)$ is the finite set $\{Q \mid P \equiv n[Q]\}/\equiv$.

LEMMA 16.

(1) For any P, $P \models \mathcal{A}' \mid \mathcal{A}'' \Leftrightarrow \exists \langle P', P'' \rangle \in T(P) . P' \models \mathcal{A}' \wedge P'' \models \mathcal{A}''$.

(2) For any $\mathcal{A}$, $\mathcal{B}$, $P \models \mathcal{A} \triangleright \mathcal{B} \Leftrightarrow \forall Q \in T(\mathcal{A} \triangleright \mathcal{B}) . Q \models \mathcal{A} \Rightarrow Q \mid P \models \mathcal{B}$.

(3) For any P, $P \models n[\mathcal{A}'] \Leftrightarrow \exists P' \in T(n, P) . P' \models \mathcal{A}'$.

PROOF. Part (2) follows at once from Theorem 1. The other parts follow easily, as in earlier work [10]. □

THEOREM 2. *Satisfaction and validity are interderivable and decidable.*

Validity is defined in terms of an infinite quantification over trees. We end with a corollary of Lemma 4 and Theorem 1, which reduces validity to a finite quantification over a computable sequence of trees. Hence, we obtain an explicit algorithm for validity.

COROLLARY 1. *Consider any formula $\mathcal{A}$. If $EQ^X_{h,w} = \{\langle P_1 \rangle_{\equiv}, \ldots, \langle P_n \rangle_{\equiv}\}$, where $(h, w) = |\mathcal{A}|^{\mathsf{hw}}$ and $X = fn(\mathcal{A}) \cup \{m\}$ for some $m \notin fn(\mathcal{A})$, then*

$$\mathbf{vld}(\mathcal{A}) \Leftrightarrow (\forall i \in 1..n . P_i \models \mathcal{A})$$

It is straightforward to implement the algorithms for satisfaction and validity suggested above. However, they are of limited practical interest, since the size of $EQ^X_{h,w}$ is not elementary (not bounded by any tower of exponentials) in the worst case. The only lower bound we know is PSPACE. Still, the algorithm terminates in a reasonable time on small formulas. Here is a selection of formulas found to be valid by our implementation.

- $(\mathbf{0} \vee p[\mathbf{0}]) \mid \neg(p[\mathbf{0}])$
- $q[\neg\mathbf{0}] \triangleright \neg(\mathbf{0})$
- $\neg((q[q[\mathbf{0}]] \mid q[\mathbf{0}])@q)$
- $(T \triangleright \neg((q[\mathbf{0}] \vee \mathbf{T}) \triangleright \mathbf{0}))@q$
- $((\mathbf{0} \vee p[\mathbf{0}])@p)@p@p$
- $(\neg(p[\mathbf{T}]) \vee \neg(q[\mathbf{T}]))@q$
- $p[\mathbf{T}] \triangleright (p[\mathbf{T}] \mid \mathbf{T})$
- $\neg(p[\mathbf{T}] \triangleright \mathbf{0})$
- $\neg(\mathbf{T} \mid (\mathbf{T} \triangleright q[\mathbf{0}])@q)$
- $(\mathbf{T} \mid (\neg(\mathbf{0}) \vee \mathbf{0})) \mid \mathbf{T}$
- $(\mathbf{T} \mid q[\mathbf{T}])@q \vee \mathbf{0}$

To see why, for example, that the formula $(\mathbf{0} \vee p[\mathbf{0}]) \mid \neg(p[\mathbf{0}])$ is valid, consider any process P. Either $P \models p[\mathbf{0}]$ or not. If so, we have $P \equiv P \mid \mathbf{0}$, and $P \models \mathbf{0} \vee p[\mathbf{0}]$ and $\mathbf{0} \models \neg(p[\mathbf{0}])$. If not, we have $P \equiv \mathbf{0} \mid P$, and $\mathbf{0} \models \mathbf{0} \vee p[\mathbf{0}]$ and $P \models \neg(p[\mathbf{0}])$. So, in either case, the process satisfies $(\mathbf{0} \vee p[\mathbf{0}]) \mid \neg(p[\mathbf{0}])$.

4. DECIDING VALIDITY BY DEDUCTION

We present a sequent calculus for our spatial logic, following the pattern of Caires and Cardelli [4]. We show the calculus to be sound and complete with respect to an interpretation in terms of the satisfaction relation, and present a complete proof procedure. Hence, we obtain an algorithm for deciding validity by deduction in the sequent calculus.

4.1 A Sequent Calculus

A *context*, Γ or Δ, is a finite multiset of entries of the form $P : \mathcal{A}$ where P is a tree and $\mathcal{A}$ is a formula. A *sequent* is a judgment $\Gamma \vdash \Delta$ where Γ and Δ are contexts. The following table states the rules for deriving sequents. The rules depend on the finite test sets $T(P)$, $T(\mathcal{A} \triangleright \mathcal{B})$, and $T(n, P)$ introduced in Section 3. All that matters for the purpose of this section is that these sets are computable and that they satisfy the properties stated in Lemma 16. Hence, this is a finitary proof system; note the form of the rules (| L), ($\triangleright$ R), and ($n[]$ L).

Rules of the Sequent Calculus:

(Id) $\dfrac{P \equiv Q}{\Gamma, P : \mathcal{A} \vdash Q : \mathcal{A}, \Delta}$ (Cut) $\dfrac{\Gamma \vdash P : \mathcal{A}, \Delta \quad \Gamma, P : \mathcal{A} \vdash \Delta}{\Gamma \vdash \Delta}$

(C L) $\dfrac{\Gamma, P : \mathcal{A}, P : \mathcal{A} \vdash \Delta}{\Gamma, P : \mathcal{A} \vdash \Delta}$ (C R) $\dfrac{\Gamma \vdash P : \mathcal{A}, P : \mathcal{A}, \Delta}{\Gamma \vdash P : \mathcal{A}, \Delta}$

(**F** L) $\dfrac{}{\Gamma, P : \mathbf{F} \vdash \Delta}$ (**F** R) $\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash P : \mathbf{F}, \Delta}$

($\wedge$ L) $\dfrac{\Gamma, P : \mathcal{A}, P : \mathcal{B} \vdash \Delta}{\Gamma, P : \mathcal{A} \wedge \mathcal{B} \vdash \Delta}$ ($\wedge$ R) $\dfrac{\Gamma \vdash P : \mathcal{A}, \Delta \quad \Gamma \vdash P : \mathcal{B}, \Delta}{\Gamma \vdash P : \mathcal{A} \wedge \mathcal{B}, \Delta}$

($\Rightarrow$ L) $\dfrac{\Gamma \vdash P : \mathcal{A}, \Delta \quad \Gamma, P : \mathcal{B} \vdash \Delta}{\Gamma, P : \mathcal{A} \Rightarrow \mathcal{B} \vdash \Delta}$ ($\Rightarrow$ R) $\dfrac{\Gamma, P : \mathcal{A} \vdash P : \mathcal{B}, \Delta}{\Gamma \vdash P : \mathcal{A} \Rightarrow \mathcal{B}, \Delta}$

(**0** L) $\dfrac{P \not\equiv \mathbf{0}}{\Gamma, P : \mathbf{0} \vdash \Delta}$ (**0** R) $\dfrac{P \equiv \mathbf{0}}{\Gamma \vdash P : \mathbf{0}, \Delta}$

(| L) $\dfrac{\forall \langle Q, R \rangle \in T(P).\ \Gamma, Q : \mathcal{A}, R : \mathcal{B} \vdash \Delta}{\Gamma, P : \mathcal{A} \mid \mathcal{B} \vdash \Delta}$

(| R) $\dfrac{\Gamma \vdash Q : \mathcal{A}, \Delta \quad \Gamma \vdash R : \mathcal{B}, \Delta \quad P \equiv Q \mid R}{\Gamma \vdash P : \mathcal{A} \mid \mathcal{B}, \Delta}$

($\triangleright$ L) $\dfrac{\Gamma \vdash Q : \mathcal{A}, \Delta \quad \Gamma, Q \mid P : \mathcal{B} \vdash \Delta}{\Gamma, P : \mathcal{A} \triangleright \mathcal{B} \vdash \Delta}$

($\triangleright$ R) $\dfrac{\forall Q \in T(\mathcal{A} \triangleright \mathcal{B}).\ \Gamma, Q : \mathcal{A} \vdash Q | P : \mathcal{B}, \Delta}{\Gamma \vdash P : \mathcal{A} \triangleright \mathcal{B}, \Delta}$

($n[]$ L) $\dfrac{\forall Q \in T(n, P).\ \Gamma, Q : \mathcal{A} \vdash \Delta}{\Gamma, P : n[\mathcal{A}] \vdash \Delta}$

($n[]$ R) $\dfrac{\Gamma \vdash Q : \mathcal{A}, \Delta \quad P \equiv n[Q]}{\Gamma \vdash P : n[\mathcal{A}], \Delta}$

($@n$ L) $\dfrac{\Gamma, n[P] : \mathcal{A} \vdash \Delta}{\Gamma, P : \mathcal{A}@n \vdash \Delta}$ ($@n$ R) $\dfrac{\Gamma \vdash n[P] : \mathcal{A}, \Delta}{\Gamma \vdash P : \mathcal{A}@n, \Delta}$

The variables Q, R in (| L) and the variable Q in ($\triangleright$ R) cannot occur free (in a formalistic reading) in Γ, P, Δ. Compare the side conditions on these rules in Caires and Cardelli [4]. Here, these are meta-level variables ranging over terms, so there is no need for such side conditions. Note that ($n[]$ L) applies also when $T(n, P)$ is empty (something that never happens for (| L)), so we can conclude, for example, $\Gamma, \mathbf{0} : n[\mathcal{A}] \vdash \Delta$. The fact that $T(n, P)$ may be empty explains also the irregular form of clause ($n[]$ R) of Lemma 18 below.

Lemma 17 (Weakening). *If $\Gamma \vdash \Delta$ is derivable, then $\Gamma, P : \mathcal{A} \vdash \Delta$ and $\Gamma \vdash P : \mathcal{A}, \Delta$ are derivable. Moreover, if there is a derivation of $\Gamma \vdash \Delta$ free of (Id), (Cut), (C L), (C R), then there are derivations of $\Gamma, P : \mathcal{A} \vdash \Delta$ and $\Gamma \vdash P : \mathcal{A}, \Delta$ free of (Id), (Cut), (C L), (C R).*

4.2 Soundness and Completeness

We make a conventional interpretation of sequents:

$$
\begin{aligned}
{}^{\wedge}[\![P_1 : \mathcal{A}_1, ..., P_n : \mathcal{A}_n]\!] &\triangleq P_1 \models \mathcal{A}_1 \wedge \ldots \wedge P_n \models \mathcal{A}_n \\
{}^{\vee}[\![Q_1 : \mathcal{B}_1, ..., Q_m : \mathcal{B}_m]\!] &\triangleq Q_1 \models \mathcal{B}_1 \vee \ldots \vee Q_m \models \mathcal{B}_m \\
[\![\Gamma \vdash \Delta]\!] &\triangleq {}^{\wedge}[\![\Gamma]\!] \Rightarrow {}^{\vee}[\![\Delta]\!]
\end{aligned}
$$

To prove soundness and completeness of the sequent calculus, we need the following two lemmas.

LEMMA 18 (VALIDITY OF ANTECEDENTS).

(F L) $[\![\Gamma, P : \mathbf{F} \vdash \Delta]\!]$

(F R) $[\![\Gamma \vdash P : \mathbf{F}, \Delta]\!]$ *iff* $[\![\Gamma \vdash \Delta]\!]$

($\wedge$ L) $[\![\Gamma, P : \mathcal{A}' \wedge \mathcal{A}'' \vdash \Delta]\!]$ *iff* $[\![\Gamma, P : \mathcal{A}', P : \mathcal{A}'' \vdash \Delta]\!]$

($\wedge$ R) $[\![\Gamma \vdash P : \mathcal{A}' \wedge \mathcal{A}'', \Delta]\!]$ *iff* $[\![\Gamma \vdash P : \mathcal{A}', \Delta]\!] \wedge [\![\Gamma \vdash P : \mathcal{A}'', \Delta]\!]$

($\vee$ L) $[\![\Gamma, P : \mathcal{A}' \vee \mathcal{A}'' \vdash \Delta]\!]$ *iff* $[\![\Gamma \vdash P : \mathcal{A}', \Delta]\!] \wedge [\![\Gamma, P : \mathcal{A}'' \vdash \Delta]\!]$

($\vee$ R) $[\![\Gamma \vdash P : \mathcal{A}' \vee \mathcal{A}'', \Delta]\!]$ *iff* $[\![\Gamma, P : \mathcal{A}' \vdash P : \mathcal{A}'', \Delta]\!]$

(0 L) $[\![\Gamma, P : \mathbf{0} \vdash \Delta]\!]$ *iff* $P \equiv \mathbf{0} \Rightarrow [\![\Gamma \vdash \Delta]\!]$

(0 R) $[\![\Gamma \vdash P : \mathbf{0}, \Delta]\!]$ *iff* $P \not\equiv \mathbf{0} \Rightarrow [\![\Gamma \vdash \Delta]\!]$

(| L) $[\![\Gamma, P : \mathcal{A}' \mid \mathcal{A}'' \vdash \Delta]\!]$ *iff* $\forall P', P''.P \equiv P' \mid P'' \Rightarrow [\![\Gamma, P' : \mathcal{A}', P'' : \mathcal{A}'' \vdash \Delta]\!]$

(| R) $[\![\Gamma \vdash P : \mathcal{A}' \mid \mathcal{A}'', \Delta]\!]$ *iff* $\exists P', P''.P \equiv P' \mid P'' \wedge [\![\Gamma \vdash P' : \mathcal{A}', \Delta]\!] \wedge [\![\Gamma \vdash P'' : \mathcal{A}'', \Delta]\!]$

($\triangleright$ L) $[\![\Gamma, P : \mathcal{A}' \triangleright \mathcal{A}'' \vdash \Delta]\!]$ *iff* $\exists P'.[\![\Gamma \vdash P' : \mathcal{A}', \Delta]\!] \wedge [\![\Gamma, P' \mid P : \mathcal{A}'' \vdash \Delta]\!]$

($\triangleright$ R) $[\![\Gamma \vdash P : \mathcal{A}' \triangleright \mathcal{A}'', \Delta]\!]$ *iff* $\forall P'.[\![\Gamma, P' : \mathcal{A}' \vdash P' \mid P : \mathcal{A}'', \Delta]\!]$

($n[]$ L) $[\![\Gamma, P : n[\mathcal{A}'] \vdash \Delta]\!]$ *iff* $\forall P'.P \equiv n[P'] \Rightarrow [\![\Gamma, P' : \mathcal{A}' \vdash \Delta]\!]$

($n[]$ R) $[\![\Gamma \vdash P : n[\mathcal{A}'], \Delta]\!]$ *iff* $(\forall P'.P \not\equiv n[P'] \wedge [\![\Gamma \vdash \Delta]\!]) \vee (\exists P'.P \equiv n[P'] \wedge [\![\Gamma \vdash P' : \mathcal{A}', \Delta]\!])$

($@n$ L) $[\![\Gamma, P : \mathcal{A}'@n \vdash \Delta]\!]$ *iff* $[\![\Gamma, n[P] : \mathcal{A}' \vdash \Delta]\!]$

($@n$ R) $[\![\Gamma \vdash P : \mathcal{A}'@n, \Delta]\!]$ *iff* $[\![\Gamma \vdash n[P] : \mathcal{A}', \Delta]\!]$

LEMMA 19 (FINITE TEST SETS).

(1) For any P there is a finite set $T(P)$ with:
$\forall P', P''.P \equiv P' \mid P'' \Rightarrow [\![\Gamma, P' : \mathcal{A}', P'' : \mathcal{A}'' \vdash \Delta]\!]$
iff $\forall \langle P', P'' \rangle \in T(P).[\![\Gamma, P' : \mathcal{A}', P'' : \mathcal{A}'' \vdash \Delta]\!]$.

(2) For any $\mathcal{A}'$, $\mathcal{A}''$, there is a finite set $T(\mathcal{A}' \triangleright \mathcal{A}'')$ with:
$\forall P'.[\![\Gamma, P' : \mathcal{A}' \vdash P' \mid P : \mathcal{A}'', \Delta]\!]$
iff $\forall P' \in T(\mathcal{A}' \triangleright \mathcal{A}'').[\![\Gamma, P' : \mathcal{A}' \vdash P' \mid P : \mathcal{A}'', \Delta]\!]$.

(3) For any P there is a finite set $T(n, P)$ with:
$\forall P'.P \equiv n[P'] \Rightarrow [\![\Gamma, P' : \mathcal{A}' \vdash \Delta]\!]$
iff $\forall P' \in T(n, P).[\![\Gamma, P' : \mathcal{A}' \vdash \Delta]\!]$.

THEOREM 3 (SOUNDNESS). *If $\Gamma \vdash \Delta$ is derivable, $[\![\Gamma \vdash \Delta]\!]$.*

PROOF. By induction on the derivation of $\Gamma \vdash \Delta$. □

THEOREM 4 (COMPLETENESS). *If $[\![\Gamma \vdash \Delta]\!]$, then $\Gamma \vdash \Delta$ has a derivation. Moreover, it has a derivation that does not use (Id), (Cut), (C L), (C R).*

PROOF. By induction on the sum of the sizes of all the formulas in $\Gamma \vdash \Delta$. The interesting cases are (| L), ($n[]$ L) and, particularly, ($\triangleright$ R), relying on Lemma 19. □

PROPOSITION 5. (ID, CUT, AND CONTRACTION ELIMINATION). *If $\Gamma \vdash \Delta$ has a derivation, then there is a derivation that does not use (Id), (Cut), (C L), (C R).*

PROOF. If $\Gamma \vdash \Delta$ is derivable in the full system, then $[\![\Gamma \vdash \Delta]\!]$ by Theorem 3 (Soundness). Then, by Theorem 4 (Completeness), $\Gamma \vdash \Delta$ has a derivation that does not use (Id), (Cut), (C L), (C R). □

By combining Theorems 2, 3, and 4 we obtain:

PROPOSITION 6 (DECIDABILITY). *It is decidable whether $\Gamma \vdash \Delta$ is derivable.*

4.3 A Complete Proof Procedure

The following theorem essentially implies Completeness, and uses Lemma 18 in a similar way, but is not quite as clean as Completeness, since it talks about an algorithm. Moreover, the cases for ($\triangleright$ L), (| R) and ($n[]$ R) are harder than in Completeness.

On the other hand, the proposition is interesting because it shows that there is a complete proof procedure that actually builds a derivation, unlike the one in Proposition 6.

LEMMA 20 (MORE ON FINITE TEST SETS).

(1) For any P there is a finite set $T(P)$ with:
$\exists P', P''.P \equiv P' \mid P'' \wedge [\![\Gamma \vdash P' : \mathcal{A}', \Delta]\!] \wedge [\![\Gamma \vdash P'' : \mathcal{A}'', \Delta]\!]$
iff $\exists \langle P', P'' \rangle \in T(P).[\![\Gamma \vdash P' : \mathcal{A}', \Delta]\!] \wedge [\![\Gamma \vdash P'' : \mathcal{A}'', \Delta]\!]$.

(2) For any $\mathcal{A}'$, $\mathcal{A}''$, there is a finite set $T(\mathcal{A}' \triangleright \mathcal{A}'')$ with:
$\exists P'.[\![\Gamma \vdash P' : \mathcal{A}', \Delta]\!] \wedge [\![\Gamma, P' \mid P : \mathcal{A}'' \vdash \Delta]\!]$
iff
$\exists P' \in T(\mathcal{A}' \triangleright \mathcal{A}'').[\![\Gamma \vdash P' : \mathcal{A}', \Delta]\!] \wedge [\![\Gamma, P' \mid P : \mathcal{A}'' \vdash \Delta]\!]$.

(3) For any P there is a finite set $T(n, P)$ with:
$\exists P'.P \equiv n[P'] \wedge [\![\Gamma \vdash P' : \mathcal{A}', \Delta]\!]$
iff $\exists P' \in T(n, P).[\![\Gamma \vdash P' : \mathcal{A}', \Delta]\!]$.

THEOREM 5 (COMPLETE PROOF PROCEDURE). *Given any $\Gamma \vdash \Delta$ there is a procedure such that: if $\neg[\![\Gamma \vdash \Delta]\!]$, then the procedure terminates with failure; if $[\![\Gamma \vdash \Delta]\!]$, then the procedure terminates with a derivation for $\Gamma \vdash \Delta$.*

PROOF. We describe the procedure, but omit the proof of correctness, which, in addition to the properties used in the proof of Theorem 4, uses also Lemma 20. The procedure picks nondeterministically any formula in the sequent to operate on. It terminates because at every recursive call it either reduces the total size, *size*, of the formulas in the sequent, or stops with success or failure.

Case *size* $= 0$, that is, the empty sequent $- \vdash -$.
The procedure terminates with failure.

Case *size* ≥ 1, left rules.

Subcase $\Gamma, P : \mathbf{F} \vdash \Delta$.
The procedure succeeds with derivation $\Gamma, P : \mathbf{F} \vdash \Delta$.

Subcase $\Gamma, P : \mathcal{A}' \wedge \mathcal{A}'' \vdash \Delta$.
The procedure recurses with $\Gamma, P : \mathcal{A}', P : \mathcal{A}'' \vdash \Delta$; if the recursion fails, it fails; if the recursion succeeds with a derivation for $\Gamma, P : \mathcal{A}', P : \mathcal{A}'' \vdash \Delta$, it produces a derivation for $\Gamma, P : \mathcal{A}' \wedge \mathcal{A}'' \vdash \Delta$ by ($\wedge$ L).

Subcase $\Gamma, P : \mathcal{A}' \Rightarrow \mathcal{A}'' \vdash \Delta$.
The procedure recurses with $\Gamma \vdash P : \mathcal{A}', \Delta$ and $\Gamma, P : \mathcal{A}'' \vdash \Delta$; if either recursion fails, the procedure fails. If the recursions succeed with derivations for $\Gamma \vdash P : \mathcal{A}', \Delta$ and $\Gamma, P : \mathcal{A}'' \vdash \Delta$ the procedure produces a derivation for $\Gamma, P : \mathcal{A}' \Rightarrow \mathcal{A}'' \vdash \Delta$ by ($\Rightarrow$ L).

Subcase $\Gamma, P : \mathbf{0} \vdash \Delta$.
If $P \not\equiv \mathbf{0}$ (a decidable test) the procedure returns with the derivation $\Gamma, P : \mathbf{0} \vdash \Delta$ by ($\mathbf{0}$ L), otherwise it recurses with $\Gamma \vdash \Delta$. If the recursion fails, it fails; if it succeeds with a derivation for $\Gamma \vdash \Delta$, it returns a derivation for $\Gamma, P : \mathbf{0} \vdash \Delta$ by weakening.

Subcase $\Gamma, P : \mathcal{A}' \mid \mathcal{A}'' \vdash \Delta$.
The procedure computes the finite set $T(P)$, and for every $\langle P', P'' \rangle$ belonging to it, it recurses with $\Gamma, P' : \mathcal{A}', P'' : \mathcal{A}'' \vdash \Delta$. If all the recursive calls succeed, the procedure builds a derivation for $\Gamma, P : \mathcal{A}' \mid \mathcal{A}'' \vdash \Delta$ by ($\mid$ L), otherwise it fails.

Subcase $\Gamma, P : \mathcal{A}' \triangleright \mathcal{A}'' \vdash \Delta$.
The procedure computes the finite set $T(\mathcal{A}' \triangleright \mathcal{A}'')$, and for every P' belonging to it, it recurses with $\Gamma \vdash P' : \mathcal{A}', \Delta$ and $\Gamma, P' \mid P : \mathcal{A}'' \vdash \Delta$. If one pair of recursive calls succeeds, the procedure builds a derivation for $\Gamma, P : \mathcal{A}' \triangleright \mathcal{A}'' \vdash \Delta$ by ($\triangleright$ L), otherwise it fails.

Subcase $\Gamma, P : n[\mathcal{A}'] \vdash \Delta$.
The procedure computes the finite set $T(n, P)$ (which may be empty). For every P' belonging to it, the procedure recurses with $\Gamma, P' : \mathcal{A}' \vdash \Delta$. If all the recursive calls succeed, the procedure builds a derivation for $\Gamma, P : n[\mathcal{A}'] \vdash \Delta$ by ($n[]$ L), otherwise it fails.

Subcase $\Gamma, P : \mathcal{A}'@n \vdash \Delta$.
The procedure recurses with $\Gamma, n[P] : \mathcal{A}' \vdash \Delta$. If the recursive call succeeds, the procedure builds a derivation for $\Gamma, P : \mathcal{A}'@n \vdash \Delta$ by ($@n$ L), otherwise it fails.

Case *size* ≥ 1, right rules.

Subcase $\Gamma \vdash P : \mathbf{F}, \Delta$.
The procedure recurses with $\Gamma \vdash \Delta$. If the recursion fails, the procedure fails. If the recursion succeeds with a derivation for $\Gamma \vdash \Delta$, the procedure returns a derivation for $\Gamma \vdash P : \mathbf{F}, \Delta$ by ($\mathbf{F}$ R).

Subcase $\Gamma \vdash P : \mathcal{A}' \wedge \mathcal{A}'', \Delta$.
The procedure recurses with $\Gamma \vdash P : \mathcal{A}', \Delta$ and $\Gamma \vdash P : \mathcal{A}'', \Delta$. If both recursive calls succeeds, the procedure builds a derivation for $\Gamma \vdash P : \mathcal{A}' \wedge \mathcal{A}'', \Delta$ by ($\wedge$ R), otherwise it fails.

Subcase $\Gamma \vdash P : \mathcal{A}' \Rightarrow \mathcal{A}'', \Delta$.
The procedure recurses with $\Gamma, P : \mathcal{A}' \vdash P : \mathcal{A}'', \Delta$. If the recursion fails, it fails; if the recursion succeeds with a derivation for $\Gamma, P : \mathcal{A}' \vdash P : \mathcal{A}'', \Delta$, it produces a derivation for $\Gamma, P : \mathcal{A}' \Rightarrow \mathcal{A}'' \vdash \Delta$ by ($\Rightarrow$ R).

Subcase $\Gamma \vdash P : \mathbf{0}, \Delta$.
If $P \equiv \mathbf{0}$ (a decidable test) the procedure returns with the derivation $\Gamma \vdash P : \mathbf{0}, \Delta$ by ($\mathbf{0}$ R), otherwise it recurses with $\Gamma \vdash \Delta$. If the recursion fails, it fails; if it succeeds with a derivation for $\Gamma \vdash \Delta$, it returns a derivation for $\Gamma \vdash P : \mathbf{0}, \Delta$ by weakening.

Subcase $\Gamma \vdash P : \mathcal{A}' \mid \mathcal{A}'', \Delta$.
The procedure computes the finite set $T(P)$, and for every $\langle P', P'' \rangle$ belonging to it, it recurses with $\Gamma \vdash P' : \mathcal{A}', \Delta$ and $\Gamma \vdash P'' : \mathcal{A}'', \Delta$. If one pair of recursive calls succeeds, the procedure builds a derivation for $\Gamma \vdash P : \mathcal{A}' \mid \mathcal{A}'', \Delta$ by ($\mid$ R), otherwise it fails.

Subcase $\Gamma \vdash P : \mathcal{A}' \triangleright \mathcal{A}'', \Delta$.
The procedure computes the finite set $T(\mathcal{A}' \triangleright \mathcal{A}'')$, and for every P' belonging to it, it recurses with $\Gamma, P' : \mathcal{A}' \vdash P' \mid P : \mathcal{A}'', \Delta$. If all these recursive calls are successful, the procedure builds a derivation for $\Gamma \vdash P : \mathcal{A}' \triangleright \mathcal{A}'', \Delta$ by ($\triangleright$ R), otherwise it fails.

Subcase $\Gamma \vdash P : n[A'], \Delta$.
The procedure computes the finite set $T(n, P)$. If $T(n, P)$ is empty, then it recurses with $\Gamma \vdash \Delta$; if the recursion fails the procedure fails, and if it succeeds with a derivation for $\Gamma \vdash \Delta$, the procedure returns a derivation for $\Gamma \vdash P : n[\mathcal{A}'], \Delta$ by weakening. If $T(n, P)$ is not empty, then for every P' belonging to it, the procedure recurses with $\Gamma \vdash P' : \mathcal{A}', \Delta$. If one of the recursive calls succeeds, the procedure builds a derivation for $\Gamma \vdash P : n[\mathcal{A}'], \Delta$ by ($n[]$ R), otherwise it fails.

Subcase $\Gamma \vdash P : \mathcal{A}'@n, \Delta$.
The procedure recurses with $\Gamma \vdash n[P] : \mathcal{A}', \Delta$. If the recursive call succeeds, the procedure builds a derivation for $\Gamma \vdash P : \mathcal{A}'@n, \Delta$ by ($@n$ R), otherwise it fails. □

By combining Lemma 4 and Theorems 3 and 4, we equate the validity problem to a particular proof search problem.

COROLLARY 2. $\mathbf{vld}(\mathcal{A})$ *if and only if* $\vdash \mathbf{0} : \mathbf{T} \triangleright \mathcal{A}$ *has a derivation.*

Hence, by Theorem 5, we obtain an algorithm for validity based on deduction.

5. A LANGUAGE FOR MANIPULATING TREES

We describe a typed λ-calculus that manipulates tree data. The type system of this calculus has, at its basis, tree types. Function types are built on top of the tree types in standard higher-order style. The tree types, however, are unusual: they are the formulas of our logic. Therefore, we can write types such as:

$$\mathbf{T} \to \neg\mathbf{0}$$
$$((\mathcal{A} \wedge \neg\mathbf{0}) \mid n[\mathcal{B}]) \to (n[\mathcal{A}] \mid \mathcal{B})$$

Logical operators can be applied only to tree types, not to higher-order types. A subtyping relation is defined between types. On tree types, subtyping is defined as validity of logical implication; that is, $\mathcal{A} <: \mathcal{B}$ means $\mathbf{vld}(\mathcal{A} \Rightarrow B)$. Subtyping is then extended to function types by the usual contravariant rule. This implies that a logical validity check is

used during static typechecking, whenever we need to check type inclusion. Tree data is manipulated via pattern matching constructs that perform "run-time type checks". Since tree types are formulas, we have the full power of the logic to express the pattern matching conditions. Those run-time type checks are executed as run-time satisfaction checks. For example, one of our matching constructs is a test to see whether the value denoted by expression t has type $\mathcal{A}$:

$$t?(x{:}\mathcal{A}).u, v$$

This construct first computes the tree P denoted by the expression t, and then performs a test $P \models \mathcal{A}$. If the test is successful, it binds P to x and executes u; otherwise it binds P to x and executes v. The variable x can be used both inside u and v, but in u it has type $\mathcal{A}$, while in v it has type $\neg\mathcal{A}$.

To summarize, our formulas are used as a very expressive type system for tree data, within a typed λ-calculus. A satisfaction algorithm is used to analyze data at run-time, and a validity algorithm is needed during static typechecking. We of course have such algorithms available, as described in previous sections, at least for ground terms and types. In absence of polymorphism or dependent types, types are in fact ground. And, at run-time, all values are ground too. As usual, the type system checks whether an open term has a (ground) type: it can do so without additional difficulties, even though the basic satisfaction test we have is for closed terms (that is, trees).

5.1 Syntax

The λ-calculus is stratified in terms of low types and high types. The low types are, in this case, just tree types, but could in general include other basic data types such as integers and names. The high types are function types over the low types. The tree types are the formulas of our logic.

The same stratification holds on terms: there are terms of low types (the trees) and terms of high types (the functions). This stratification is not reflected in the syntax, essentially because variables can hold high or low values, but it is reflected in the operational semantics.

SYNTAX:

$\mathcal{F}, \mathcal{G}, \mathcal{H} ::=$	High Types
$\mathcal{A}$	tree types (formulas of the logic)
$\mathcal{F} \to \mathcal{G}$	function types
$t, u, v ::=$	terms
$\mathbf{0}$	void
$n[t]$	location
$t \mid u$	composition
$t?n[x{:}\mathcal{A}].u$	location match
$t?(x{:}\mathcal{A} \mid y{:}\mathcal{B}).u$	composition match
$t?(x{:}\mathcal{A}).u, v$	tree type match
x	variable
$\lambda x{:}\mathcal{F}.t$	function
$t(u)$	application

The syntax of terms provides: a standard λ-calculus fragment, the three basic tree constructors, and three matching operators for analyzing tree data. The tree type match construct performs a run-time check to see whether a tree matches a given formula. Then one needs other constructs to decompose the trees: a composition match splits a tree in two components, and a location match strips an edge from a tree. A zero match is redundant because of the tree type match construct.

These multiple matching constructs are designed to simplify the operational semantics and the type rules. In practice, one would use a single case statement over the structure of trees; this can be easily translated to the given matching constructs. For example:

Case Statement

case t *of*	analyze t
$\mathbf{0}.u_1$,	if $t \equiv \mathbf{0}$, run u_1, else
$n[x{:}\mathcal{A}].u_2$,	if $t \equiv n[P]$ and $P \models A$, bind P to x and run u_2, else
$(x{:}\mathcal{A} \mid y{:}\mathcal{B}).u_3$,	if $t \equiv P \mid Q$ and $P \models \mathcal{A}$, $Q \models \mathcal{B}$, bind P to x, Q to y and run u_3,
else u_4	else run u_4
$\triangleq$	can be translated as:

$t?(z_1{:}\mathbf{0}).u_1,$
$t?(z_2{:}n[\mathcal{A}]).z_2?n[x{:}\mathcal{A}].u_2,$
$t?(z_3{:}\mathcal{A} \mid \mathcal{B}).z_3?(x{:}\mathcal{A} \mid y{:}\mathcal{B}).u_3,$
u_4

Further, one may want to allow complex nested patterns, that can be translated to nested uses of the given matching constructs.

5.2 Values

Programs in the syntax of the previous section produce values; either tree values or function values (that is, closures). Over the tree values we define the usual structural congruence $\equiv$; the matching constructs of the language are not able to distinguish between structurally congruent trees. The function values are triples of a term t with respect to an input variable x (that is, essentially $\lambda x.t$) and a stack for free variables ρ. A stack ρ is a list of bindings x, F of variables to values, with possible repetitions of the variables.

VALUES:

$F, G, H ::=$	High Values
P	tree values
$\langle \rho, x, t \rangle$	function values
ρ is a list of x, F pairs	Stacks
$\rho[x{\leftarrow}F]$ is ρ plus an x, F pair at the end	
$\rho(x)$ is the last F associated with x (if any)	

5.3 Operational Semantics

The operational semantics is given by a relation $t \Downarrow_\rho F$ between terms t, stacks ρ, and values F, meaning that t can evaluate to F on stack ρ. The semantics makes use of the satisfaction relation $P \models \mathcal{A}$ from Section 2. We use, for example, $t \Downarrow_\rho P$ to indicate that t evaluates to a tree value P. We use $t \Downarrow_\rho \equiv P$ as an abbreviation for $t \Downarrow_\rho Q$ and $Q \equiv P$, for some Q.

OPERATIONAL SEMANTICS

(Red $\mathbf{0}$)	(Red $\mid$)	(Red $n[]$)
	$\dfrac{t \Downarrow_\rho P \quad u \Downarrow_\rho Q}{t \mid u \Downarrow_\rho P \mid Q}$	$\dfrac{t \Downarrow_\rho P}{n[t] \Downarrow_\rho n[P]}$
$\mathbf{0} \Downarrow_\rho \mathbf{0}$		

(Red $?n[]$)
$$\frac{t \Downarrow_\rho \equiv n[P] \quad P \models \mathcal{A} \quad u \Downarrow_{\rho[x \leftarrow P]} F}{t?n[x{:}A].u \Downarrow_\rho F}$$

(Red $?|$)
$$\frac{t \Downarrow_\rho \equiv P' \mid P'' \quad P' \models \mathcal{A} \quad P'' \models \mathcal{B} \quad u \Downarrow_{\rho[x \leftarrow P'][y \leftarrow P'']} F}{t?(x{:}\mathcal{A} \mid y{:}\mathcal{B}).u \Downarrow_\rho F}$$

(Red $?1$)
$$\frac{t \Downarrow_\rho P \quad P \models \mathcal{A} \quad u \Downarrow_{\rho[x \leftarrow P]} F}{t?(x{:}\mathcal{A}).u, v \Downarrow_\rho F}$$

(Red $?2$)
$$\frac{t \Downarrow_\rho P \quad P \models \neg\mathcal{A} \quad v \Downarrow_{\rho[x \leftarrow P]} F}{t?(x{:}\mathcal{A}).u, v \Downarrow_\rho F}$$

(Red Var) $\dfrac{x \in dom(\rho)}{x \Downarrow_\rho \rho(x)}$ (Red Lam) $\dfrac{}{\lambda x{:}\mathcal{F}.t \Downarrow_\rho \langle \rho, x, t \rangle}$

(Red App)
$$\frac{t \Downarrow_\rho \langle \rho', x, t' \rangle \quad u \Downarrow_\rho G \quad t' \Downarrow_{\rho'[x \leftarrow G]} H}{t(u) \Downarrow_\rho H}$$

5.4 Type System

The type system uses environments E, which are lists of associations $x{:}\mathcal{F}$ of unique variables and their types. We indicate by $dom(E)$ the set of variables defined in E, by $E, x{:}\mathcal{F}$ the extension of E with a new association $x{:}\mathcal{F}$ (provided that $x \notin dom(E)$), and by $E(x)$ the type associated with x in E (provided that $x \in dom(E)$).

The judgments are:

Judgments:

$\mathcal{F} <: \mathcal{G}$	$\mathcal{F}$ is a subtype of $\mathcal{G}$
$E \vdash \diamond$	E is well-formed
$E \vdash t : \mathcal{F}$	t has type $\mathcal{F}$ in E

A validity test is used in the (Sub Tree) rule.

Type Rules:

(Env $\emptyset$) $\dfrac{}{\emptyset \vdash \diamond}$ (Env x) $\dfrac{E \vdash \diamond \quad x \notin dom(E)}{E, x{:}\mathcal{F} \vdash \diamond}$

(Term $\mathbf{0}$) $\dfrac{E \vdash \diamond}{E \vdash \mathbf{0} : \mathbf{0}}$ (Term $|$) $\dfrac{E \vdash t : \mathcal{A} \quad E \vdash u : \mathcal{B}}{E \vdash t \mid u : \mathcal{A} \mid \mathcal{B}}$

(Term $n[]$) $\dfrac{E \vdash t : \mathcal{A}}{E \vdash n[t] : n[\mathcal{A}]}$ (Term $?|$) $\dfrac{E \vdash t : \mathcal{A} \mid \mathcal{B} \quad E, x{:}\mathcal{A}, y{:}\mathcal{B} \vdash u : \mathcal{F}}{E \vdash t?(x{:}\mathcal{A} \mid y{:}\mathcal{B}).u : \mathcal{F}}$

(Term $?n[]$)
$$\frac{E \vdash t : n[\mathcal{A}] \quad E, x{:}\mathcal{A} \vdash u : \mathcal{F}}{E \vdash t?n[x{:}\mathcal{A}].u : \mathcal{F}}$$

(Term $?$)
$$\frac{E \vdash t : \mathcal{B} \quad E, x{:}\mathcal{A} \vdash u : \mathcal{F} \quad E, x{:}\neg\mathcal{A} \vdash v : \mathcal{F}}{E \vdash t?(x{:}\mathcal{A}).u, v : \mathcal{F}}$$

(Term Var) $\dfrac{E \vdash \diamond}{E \vdash x : E(x)}$ (Term Lam) $\dfrac{E, x{:}\mathcal{F} \vdash t : \mathcal{G}}{E \vdash \lambda x{:}\mathcal{F}.t : \mathcal{F} \to \mathcal{G}}$

(Term App) $\dfrac{E \vdash t : \mathcal{F} \to \mathcal{G} \quad E \vdash u : \mathcal{F}}{E \vdash t(u) : \mathcal{G}}$ (Subsumption) $\dfrac{E \vdash t : \mathcal{F} \quad \mathcal{F} <: \mathcal{G}}{E \vdash t : \mathcal{G}}$

(Sub Tree) $\dfrac{\mathbf{vld}(\mathcal{A} \Rightarrow \mathcal{B})}{A <: B}$ (Sub $\to$) $\dfrac{\mathcal{F}' <: \mathcal{F} \quad \mathcal{G} <: \mathcal{G}'}{\mathcal{F} \to \mathcal{G} <: \mathcal{F}' \to \mathcal{G}'}$

Since types are ground, we do not need reflexivity and transitivity rules for subtyping. Reflexivity for the base case derives from $\mathbf{vld}(\mathcal{A} \Rightarrow \mathcal{A})$.

In order to derive some basic results, we need to define a satisfaction relation between values and types. Over tree types, this is just the satisfaction relation of Section 2, $P \models \mathcal{A}$. This is then generalized to closures by saying that $\langle \rho, x, t \rangle \models \mathcal{F} \to \mathcal{G}$ if for every $F \models \mathcal{F}$, the result G of evaluating t with F bound to x on stack ρ, is such that $G \models \mathcal{G}$. Moreover we say that a stack satisfies an environment, $\rho \models E$, if $\rho(x) \models E(x)$ for all the variables defined in E.

Satisfaction:

$P \models \mathcal{A}$	as in Section 2
$H \models \mathcal{F} \to \mathcal{G}$ iff $H = \langle \rho, x, t \rangle$ and $\forall F, G.(F \models \mathcal{F} \wedge t \Downarrow_{\rho[x \leftarrow F]} G) \Rightarrow G \models \mathcal{G}$	
$\rho \models E$ iff $\forall x \in dom(E).\rho(x) \models E(x)$	

Proposition 7 (Subsumption). *If $\mathcal{F} <: \mathcal{G}$ and $H \models \mathcal{F}$ then $H \models \mathcal{G}$.*

Proof. Induction on the derivation of $\mathcal{F} <: \mathcal{G}$.

(Sub Tree) We have $\mathbf{vld}(\mathcal{A} \Rightarrow \mathcal{B})$ and $H \models \mathcal{A}$; hence H is a tree value, and $H \models \mathcal{B}$ by definition of **vld**.

(Sub $\to$) We have $\mathcal{F}' <: \mathcal{F}$ and $\mathcal{G} <: \mathcal{G}'$, and $H \models \mathcal{F} \to \mathcal{G}$. By definition, $H = \langle \rho, x, t \rangle$ and $\forall F, G.(F \models \mathcal{F} \wedge t \Downarrow_{\rho[x \leftarrow F]} G) \Rightarrow G \models \mathcal{G}$. Take any $F \models \mathcal{F}'$; by Ind Hyp $F \models \mathcal{F}$. Assume $t \Downarrow_{\rho[x \leftarrow F]} G$, then $G \models \mathcal{G}$, and by Ind Hyp $G \models \mathcal{G}'$. We have shown that $\forall F, G.(F \models \mathcal{F}' \wedge t \Downarrow_{\rho[x \leftarrow F]} G) \Rightarrow G \models \mathcal{G}'$. That is, we have shown that $\langle \rho, x, t \rangle \models \mathcal{F}' \to \mathcal{G}'$. □

Proposition 8 (Subject Reduction). *If $E \vdash t : \mathcal{F}$ and $\rho \models E$ and $t \Downarrow_\rho F$, then $F \models \mathcal{F}$.*

Proof. Induction on the derivation of $E \vdash t : \mathcal{F}$.

(Term 0) We have $E \vdash \mathbf{0} : \mathbf{0}$ and $\rho \models E$ and $\mathbf{0} \Downarrow_\rho \mathbf{0}$. By definition, $\mathbf{0} \models \mathbf{0}$.

(Term $n[]$) We have $E \vdash n[t] : n[\mathcal{A}]$ and $\rho \models E$ and $n[t] \Downarrow_\rho F$. We must have from (Term $n[]$) that $E \vdash t : \mathcal{A}$. We must have from (Red $n[]$) that $F = n[P]$ and $t \Downarrow_\rho P$. By Ind Hyp, $P \models \mathcal{A}$, hence by definition $n[P] \models n[\mathcal{A}]$.

(Term $|$) We have $E \vdash t \mid u : \mathcal{A} \mid \mathcal{B}$ and $\rho \models E$ and $t \mid u \Downarrow_\rho F$. We must have from (Term $|$) that $E \vdash t : \mathcal{A}$ and $E \vdash u : \mathcal{B}$. We must have from (Red $|$) that $F = P \mid Q$ and $t \Downarrow_\rho P$ and $u \Downarrow_\rho Q$. By Ind Hyp, $P \models \mathcal{A}$ and $Q \models \mathcal{B}$, hence by definition $t \mid u \models \mathcal{A} \mid \mathcal{B}$.

(Term $?n[]$**)** We have $E \vdash t?n[x{:}\mathcal{A}].u : \mathcal{F}$ and $\rho \models E$ and $t?n[x{:}\mathcal{A}].u \Downarrow_\rho F$. We must have from (Term $?n[]$) that $E \vdash t : n[\mathcal{A}]$ and $E, x{:}\mathcal{A} \vdash u : \mathcal{F}$. We must have from (Red $?n[]$) that $t \Downarrow_\rho \equiv n[P]$ and $P \models \mathcal{A}$ and $u \Downarrow_{\rho[x \leftarrow P]} F$. We have that $\rho[x \leftarrow P] \models E, x{:}\mathcal{A}$. By Ind Hyp $E, x{:}\mathcal{A} \vdash u : \mathcal{F}$ and $\rho[x \leftarrow P] \models E, x{:}\mathcal{A}$ and $u \Downarrow_{\rho[x \leftarrow P]} F$ implies $F \models \mathcal{F}$.

(Term $?|$**)** We have $t?(x{:}\mathcal{A} \mid y{:}\mathcal{B}).u : \mathcal{F}$ and $\rho \models E$ and $t?(x{:}A \mid y{:}B).u \Downarrow_\rho F$. We must have from (Term $?|$) that $E \vdash t : \mathcal{A} \mid \mathcal{B}$ and $E, x{:}\mathcal{A}, y{:}\mathcal{B} \vdash u : \mathcal{F}$. We must have from (Red $?|$) that $t \Downarrow_\rho \equiv P' \mid P''$ and $P' \models \mathcal{A}$ and $P'' \models \mathcal{B}$ and $u \Downarrow_{\rho[x \leftarrow P'][y \leftarrow P'']} F$. We have that $\rho[x \leftarrow P][y \leftarrow Q] \models E, x{:}\mathcal{A}, y{:}\mathcal{B}$. By Ind Hyp $E, x{:}\mathcal{A}, y{:}\mathcal{B} \vdash u : \mathcal{F}$ and $\rho[x \leftarrow P][y \leftarrow Q] \models E, x{:}\mathcal{A}, y{:}\mathcal{B}$ and $u \Downarrow_{\rho[x \leftarrow P'][y \leftarrow P'']} F$ implies $F \models \mathcal{F}$.

(Term $?$**)** We have $E \vdash t?(x{:}\mathcal{A}).u, v : \mathcal{F}$ and $\rho \models E$ and $t?(x{:}\mathcal{A}).u, v \Downarrow_\rho F$. We must have from (Term $?$) that $E \vdash t : \mathcal{B}$ and $E, x{:}\mathcal{A} \vdash u : \mathcal{F}$ and $E, x{:}\neg\mathcal{A} \vdash v : \mathcal{F}$. The reduction may come from (Red ?1); then $t \Downarrow_\rho P$ and $P \models \mathcal{A}$ and $u \Downarrow_{\rho[x \leftarrow P]} F$. We have that $\rho[x \leftarrow P] \models E, x{:}\mathcal{A}$. By Ind Hyp $E, x{:}\mathcal{A} \vdash u : \mathcal{F}$ and $\rho[x \leftarrow P] \models E, x{:}\mathcal{A}$ and $u \Downarrow_{\rho[x \leftarrow P]} F$ implies $F \models \mathcal{F}$. Else the reduction must come from (Red ?2); then $t \Downarrow_\rho P$ and $P \models \neg\mathcal{A}$ and $v \Downarrow_{\rho[x \leftarrow P]} F$. We have that $\rho[x \leftarrow P] \models E, x{:}\neg\mathcal{A}$. By Ind Hyp $E, x{:}\neg\mathcal{A} \vdash v : \mathcal{F}$ and $\rho[x \leftarrow P] \models E, x{:}\neg\mathcal{A}$ and $v \Downarrow_{\rho[x \leftarrow P]} F$ implies $F \models \mathcal{F}$.

(Term Var) We have $E \vdash x : E(x)$ and $\rho \models E$ and $x \Downarrow_\rho F$. We must have from (Red Var) that $F = \rho(x)$. Since $\rho \models E$, we have that $\rho(x) \models E(x)$, that is, $F \models E(x)$.

(Term Lam) We have $E \vdash \lambda x{:}\mathcal{F}.t : \mathcal{F} \to \mathcal{G}$ and $\rho \models E$ and $\lambda x{:}\mathcal{F}.t \Downarrow_\rho F$. We must have from (Red Lam) that $F = \langle \rho, x, t \rangle$. We need to show that $\langle \rho, x, t \rangle \models \mathcal{F} \to \mathcal{G}$, that is, that $\forall F, G.(F \models \mathcal{F} \wedge t \Downarrow_{\rho[x \leftarrow F]} G) \Rightarrow G \models \mathcal{G}$. Take any $F \models \mathcal{F}$, then $\rho[x \leftarrow F] \models E, x{:}\mathcal{F}$. Assuming that $t \Downarrow_{\rho[x \leftarrow F]} G$ we need to show that $G \models \mathcal{G}$. We must have from (Term Lam) that $E, x{:}\mathcal{F} \vdash t : \mathcal{G}$. By Ind Hyp if $\rho[x \leftarrow F] \models E, x{:}\mathcal{F}$ and $t \Downarrow_{\rho[x \leftarrow F]} G$, then $G \models \mathcal{G}$.

(Term App) We have $E \vdash t(u) : \mathcal{F}$ and $\rho \models E$ and $t(u) \Downarrow_\rho F$. We must have from (Term App) that $E \vdash t : \mathcal{G} \to \mathcal{F}$ and $E \vdash u : \mathcal{G}$. We must have from (Red App) that $t \Downarrow_\rho \langle \rho', x, t' \rangle$ and $u \Downarrow_\rho \mathcal{G}$ and $t' \Downarrow_{\rho'[x \leftarrow G]} F$. By Ind Hyp if $E \vdash t : \mathcal{G} \to \mathcal{F}$ and $\rho \models E$ and $t \Downarrow_\rho \langle \rho', x, t' \rangle$ then $\langle \rho', x, t' \rangle \models \mathcal{G} \to \mathcal{F}$. That means that $\forall G', F'.(G' \models \mathcal{G} \wedge t' \Downarrow_{\rho'[x \leftarrow G']} F') \Rightarrow F' \models \mathcal{F}$. By Ind Hyp if $E \vdash u : \mathcal{G}$ and $\rho \models E$ and $u \Downarrow_\rho G$ then $G \models \mathcal{G}$. Hence, by taking $G' = G$ and $F' = F$, we conclude $F \models \mathcal{F}$.

(Subsumption) We have $E \vdash t : \mathcal{F}$ and $\rho \models E$ and $t \Downarrow_\rho F$. We must have from (Subsumption) that $E \vdash t : \mathcal{G}$ and $\mathcal{G} <: \mathcal{F}$. By Ind Hyp, $F \models \mathcal{G}$. By Proposition 7, $F \models \mathcal{F}$. □

5.5 Examples

The following program inspects an arbitrary tree (that is, anything of type $\mathbf{T}$). If the tree is $\mathbf{0}$ it returns the tree $a[\mathbf{0}]$, otherwise it returns the input tree. Hence the result is never $\mathbf{0}$, and the result type can be set to $\neg\mathbf{0}$.

$$\lambda x{:}\mathbf{T}.x?(y{:}\mathbf{0}).a[\mathbf{0}], y : \mathbf{T} \to \neg\mathbf{0}$$

Here is a (truncated) typing derivation; note the use of the subsumption rule to determine that $a[\mathbf{0}] <: \neg\mathbf{0}$. Each judgment is derived from the lines above it at the next level of indentation.

$$\begin{array}{rl}
E, x{:}\mathbf{T} \vdash x{:}\mathbf{T} & \text{(Term Var)} \\
E, x{:}\mathbf{T}, y{:}\mathbf{0} \vdash \mathbf{0} : \mathbf{0} & \text{(Term } \mathbf{0}\text{)} \\
E, x{:}\mathbf{T}, y{:}\mathbf{0} \vdash a[\mathbf{0}] : a[\mathbf{0}] & \text{(Term } n[]\text{)} \\
a[\mathbf{0}] <: \neg\mathbf{0} & \text{(Sub Tree)} \\
E, x{:}\mathbf{T}, y{:}\mathbf{0} \vdash a[\mathbf{0}] : \neg\mathbf{0} & \text{(Subsumption)} \\
E, x{:}\mathbf{T}, y{:}\neg\mathbf{0} \vdash y : \neg\mathbf{0} & \text{(Term Var)} \\
E, x{:}\mathbf{T} \vdash x?(y{:}\mathbf{0}).a[\mathbf{0}], y : \neg\mathbf{0} & \text{(Term ?)} \\
E \vdash \lambda x{:}\mathbf{T}.x?(y{:}\mathbf{0}).a[\mathbf{0}], y : \mathbf{T} \to \neg\mathbf{0} & \text{(Term Lam)}
\end{array}$$

6. CONCLUSIONS

This paper concerns a propositional spatial logic for finite edge-labelled trees. The spatial modalities are composition $\mathcal{A} \mid \mathcal{B}$, guarantee $\mathcal{A} \triangleright \mathcal{B}$, void $\mathbf{0}$, location $n[\mathcal{A}]$, and placement $\mathcal{A}@n$. There are two main results. First, satisfaction and validity are equivalent and decidable. Second, there is a sound and complete proof system for validity. We know of no previous algorithms for satisfaction or validity in the presence of the guarantee operator.

The spatial logic of this paper is a fragment of the ambient logic introduced by Cardelli and Gordon [10, 11]. Model checking algorithms for various fragments without guarantee have been proposed [12, 13]. Lugiez and Dal Zilio [20] show decidability of the satisfiability problem for another fragment of the ambient logic, but without guarantee; their techniques are based on tree automata.

Validity for some other propositional substructural logics turns out to be undecidable. Urquhart proves undecidability for propositional relevant logic [23]. Lincoln, Mitchell, Scedrov, and Shankar [19] prove undecidability for both propositional linear logic and propositional intuitionistic linear logic. See Cardelli and Gordon [10] for a detailed discussion of the differences between the ambient logic and relevant and linear logics.

Calcagno, Yang, and O'Hearn [7] show decidability of validity in a propositional substructural logic for reasoning about heaps. The proof in this paper is an adaptation of their proof technique.

We briefly consider the prospects of extending our results:

- Charatonik and Talbot [13] show that validity becomes undecidable in a spatial logic with name quantification. (Their result depends only on the presence of propositional logic, $\mathbf{0}$, $n[\mathcal{A}]$, $\mathcal{A} \mid \mathcal{B}$, and $\forall x.\mathcal{A}$.)

- Caires and Monteiro [5] and Cardelli and Gordon [11] introduce logical modalities to deal with fresh names. A prerequisite of studying these operators would be to enrich our tree model with fresh names.

We obtain only preliminary results about the complexity of validity for our logic from the constructions of this paper. It is easy to show that PSPACE is a lower-bound, by reduction from the Quantified Boolean Variables problem. However, there is still a significant gap between PSPACE and the complexity of our algorithm: it is easy to see that the number of equivalence classes is not elementary (not bounded by a tower of exponentials) in the size parameter. We can obtain a higher complexity lower-bound for an extension of our logic with a Kleene star operator, $\mathcal{A}^*$ (zero

or more copies of $\mathcal{A}$ in parallel). The extended logic can encode Presburger arithmetic, whose satisfiability problem is known to be complete for a class between double and triple exponential time. However, our algorithm cannot be trivially extended: there is a formula A^* that would invalidate our results when assigned any finite size.

Finally, building on some of the results of this paper, Cohen [14] proposes improvements to the algorithms for satisfaction and validity of Section 3. He studies a multiset logic, able to encode our logic, and including Kleene star. He exploits a symbolic representation of multisets to show that the validity problem for this logic is PSPACE-complete, an improvement on our upper-bound. Moreover, he describes a model checking algorithm that runs in time linear in the size of the model.

Acknowledgements. Ernie Cohen, Silvano Dal Zilio, Philippa Gardner, and Etienne Lozes made useful comments.

7. REFERENCES

[1] Extensible markup language. *http://www.w3.org/XML/*.

[2] P. Buneman. Semistructured data. In *16th ACM Symposium on Principles of Database Systems (PODS'97)*, 1997.

[3] L. Caires and L. Cardelli. A spatial logic for concurrency (Part I). In *Theoretical Aspects of Computer Software (TACS 2001)*, volume 2215 of *Lecture Notes in Computer Science*, pages 1–37. Springer, 2001.

[4] L. Caires and L. Cardelli. A spatial logic for concurrency (Part II). In *CONCUR 2002—Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 209–225. Springer, 2002.

[5] L. Caires and L. Monteiro. Verifiable and executable logic specifications of concurrent objects in L_π. In *Proceedings of the 7th European Symposium on Programming (ESOP'99)*, volume 1381 of *Lecture Notes in Computer Science*, pages 42–56. Springer, 1998.

[6] C. Calcagno, L. Cardelli, and A. D. Gordon. Deciding validity in a spatial logic for trees. Technical Report MSR–TR–2002–113, Microsoft Research, 2002.

[7] 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 (FSTTCS'01)*, volume 2245 of *Lecture Notes in Computer Science*, pages 108–119. Springer, 2001.

[8] L. Cardelli, P. Gardner, and G. Ghelli. A spatial logic for querying graphs. In *Automata, Languages and Programming (ICALP'02)*, volume 2380 of *Lecture Notes in Computer Science*, pages 597–610. Springer, 2002.

[9] L. Cardelli and G. Ghelli. A query language based on the ambient logic. In *Proceedings of the 9th European Symposium on Programming (ESOP'01)*, volume 2028 of *LNCS*, pages 1–22. Springer, 2001.

[10] L. Cardelli and A. D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *27th ACM Symposium on Principles of Programming Languages (POPL'00)*, pages 365–377, 2000.

[11] L. Cardelli and A. D. Gordon. Logical properties of name restriction. In *Proceedings of the 5th International Conference on Typed Lambda Calculi and Applications (TLCA'01)*, volume 2044 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2001.

[12] W. Charatonik, S. Dal Zilio, A. D. Gordon, S. Mukhopadhyay, and J.-M. Talbot. The complexity of model checking mobile ambients. In *Proceedings FoSSaCS'01*, volume 2030 of *LNCS*, pages 152–167. Springer, 2001. An extended version appears as Technical Report MSR–TR–2001–03, Microsoft Research, 2001.

[13] W. Charatonik and J.-M. Talbot. The decidability of model checking mobile ambients. In *Proceedings of the 15th Annual Conference of the European Association for Computer Science Logic*, volume 2142 of *LNCS*, pages 339–354. Springer, 2001.

[14] E. Cohen. Validity and model checking for logics of finite multisets. Draft, Microsoft Research, 2002.

[15] D. Hirschkoff, E. Lozes, and D. Sangiorgi. Separability, expressiveness, and decidability in the ambient logic. In *Logic in Computer Science (LICS'02)*, pages 423–432. IEEE, 2002.

[16] H. Hosoya and B. C. Pierce. XDuce: A typed XML processing language. In *Third International Workshop on the Web and Databases (WebDB2000)*, volume 1997 of *Lecture Notes in Computer Science*, pages 226–244. Springer, 2000.

[17] H. Hosoya and B. C. Pierce. Regular expression pattern matching for XML. In *28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 67–80, 2001.

[18] S. Ishtiaq and P. W. O'Hearn. BI as an assertion language for mutable data structures. In *28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 14–26, 2001.

[19] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56:239–311, 1992.

[20] D. Lugiez and S. Dal Zilio. Multitrees automata, Presburger's constraints and tree logics. Laboratoire d'Informatique Fondamentale, CNRS and Université de Provence, 2002.

[21] P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *Computer Science Logic (CSL'01)*, volume 2142 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2001.

[22] J. C. Reynolds. Separation logic: a logic for shared mutable data structures. In *Logic in Computer Science (LICS'02)*, pages 55–74. IEEE, 2002.

[23] A. Urquhart. The undecidability of entailment and relevant implication. *Journal of Symbolic Logic*, 45:1059–1073, 1984.