

Deciding Validity in a Spatial Logic for Trees

Cristiano Calcagno Luca Cardelli
Andrew D. Gordon

December 2002

Technical Report
MSR-TR-2002-113

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Publication History

A portion of this work appears in the proceedings of the *ACM SIGPLAN Workshop on Types in Language Design and Implementation*, New Orleans, January 18, 2003.

Affiliation

Cristiano Calcagno is with Queen Mary, University of London. The authors completed part of this work at Queen Mary, University of London and part at Microsoft Research, Cambridge.

Deciding Validity in a Spatial Logic for Trees

Cristiano Calcagno Luca Cardelli Andrew D. Gordon

December 2002

Abstract

We consider a propositional spatial logic for finite trees. The logic includes $\mathcal{A} | \mathcal{B}$ (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.

Contents

1	Introduction	1
2	Ground Propositional Spatial Logic (Review)	2
2.1	Edge-Labelled Finite Trees	3
2.2	Logical Formulas and Satisfaction	3
2.3	Validity of a Formula	4
3	Deciding Validity by Model Checking	5
3.1	Bounding the Names to Consider	6
3.2	Bounding the Sizes to Consider	7
3.3	Enumerating Equivalence Classes	13
4	Deciding Validity by Deduction	17
4.1	A Sequent Calculus	17
4.2	Soundness and Completeness	19
4.3	A Complete Proof Procedure	21
5	A Language for Manipulating Trees	24
5.1	Syntax	25
5.2	Values	26
5.3	Operational Semantics	26
5.4	Type System	27
5.5	Examples	30
6	Conclusions	30

1 Introduction

Due to the growing popularity of semistructured data [Bun97], and particularly XML [XML], 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 [HP00].

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 [HP01], 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 [CM98], heaps [Rey02, IO01, ORY01], trees [CG01a], graphs [CGG02], and also process calculi such as the π -calculus [CC01, CC02] and the ambient calculus [CG00, CG01b]. 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, \dots, n_k , leading to k subtrees P_1, \dots, 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* $\mathcal{A} | \mathcal{B}$ (satisfied by composite trees $P | Q$ 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.

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, Hirschhoff, Lozes, and Sangiorgi [HLS02] 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 \mathbb{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 $X \subseteq \mathbb{N}$, let $\text{Tree}_X \triangleq \{P \mid fn(P) \subseteq X\}$.

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 $P \in \text{Tree}_X$ and $P \equiv Q$ then $Q \in \text{Tree}_X$.*

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: $P \models \mathcal{A}$

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

A basic property is that structural congruence preserves satisfaction:

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

Proof An easy induction on the structure of \mathcal{A} . □

It is useful to know that every tree P has a characteristic formula \underline{P} . Let $\mathbf{0} \triangleq \mathbf{0}$, $\underline{P \mid Q} \triangleq \underline{P} \mid \underline{Q}$, and $\underline{n[P]} \triangleq n[\underline{P}]$. 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 $Q \equiv P$.*

Proof An easy induction on the structure of 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 [CG00], 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 $\mathbf{0} \models \mathbf{T} \triangleright \mathcal{A}$.

Proof With appeal to Lemma 2, we get: $\mathbf{vld}(\mathcal{A}) \Leftrightarrow (\forall P.P \models \mathcal{A}) \Leftrightarrow (\forall P.P \models \mathbf{T} \Rightarrow P \mid \mathbf{0} \models \mathcal{A}) \Leftrightarrow \mathbf{0} \models \mathbf{T} \triangleright \mathcal{A}$. \square

Lemma 5 (Satisfaction from Validity) $P \models \mathcal{A}$ if and only if $\mathbf{vld}(P \Rightarrow \mathcal{A})$.

Proof With appeal to Lemmas 2 and 3, we get: $\mathbf{vld}(P \Rightarrow \mathcal{A}) \Leftrightarrow (\forall Q.Q \models P \Rightarrow Q \models \mathcal{A}) \Leftrightarrow (\forall Q.Q \equiv P \Rightarrow Q \models \mathcal{A}) \Leftrightarrow P \models \mathcal{A}$. \square

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:*

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

Proof

$$\begin{aligned} P\{m \leftarrow m'\} &\models n[Q] \\ \Leftrightarrow \exists m'', P'. P &\equiv m''[P'] \wedge m''\{m \leftarrow m'\} = n \wedge P'\{m \leftarrow m'\} \equiv Q \\ \Leftrightarrow \exists P'. P &\equiv n[P'] \wedge P'\{m \leftarrow m'\} \equiv Q \end{aligned}$$

□

Lemma 7

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

Proof Immediate since substitution preserves the structure of trees. □

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

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

Proof By induction on the structure of \mathcal{A} . We only consider the interesting cases.

Case $\mathcal{A} \mid \mathcal{B}$. We have $m, m' \notin \text{fn}(\mathcal{A}) \cup \text{fn}(\mathcal{B})$. With appeal to Lemma 2 and Lemma 7, and the induction hypothesis, we calculate:

$$\begin{aligned} P \models \mathcal{A} \mid \mathcal{B} & \\ \Leftrightarrow \exists P', P''. P &\equiv P' \mid P'' \wedge P' \models \mathcal{A} \wedge P'' \models \mathcal{B} \\ \Leftrightarrow \exists P', P''. P &\equiv P' \mid P'' \wedge P'\{m \leftarrow m'\} \models \mathcal{A} \wedge P''\{m \leftarrow m'\} \models \mathcal{B} \\ \Leftrightarrow \exists P', P'', Q', Q''. P &\equiv P' \mid P'' \wedge Q' \equiv P'\{m \leftarrow m'\} \wedge \\ &Q'' \equiv P''\{m \leftarrow m'\} \wedge Q' \models \mathcal{A} \wedge Q'' \models \mathcal{B} \\ \Leftrightarrow \exists Q', Q''. P\{m \leftarrow m'\} &\equiv Q' \mid Q'' \wedge Q' \models \mathcal{A} \wedge Q'' \models \mathcal{B} \\ \Leftrightarrow P\{m \leftarrow m'\} &\models \mathcal{A} \mid \mathcal{B} \end{aligned}$$

Case $\mathcal{A} \triangleright \mathcal{B}$. We have $m, m' \notin \text{fn}(\mathcal{A}) \cup \text{fn}(\mathcal{B})$. With appeal to the induction hypothesis, we calculate:

$$\begin{aligned} P \models \mathcal{A} \triangleright \mathcal{B} &\Leftrightarrow \forall Q. Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B} \\ &\Leftrightarrow \forall Q. Q \models \mathcal{A} \Rightarrow (P \mid Q)\{m \leftarrow m'\} \models \mathcal{B} \\ &\Leftrightarrow \forall Q. Q \models \mathcal{A} \Rightarrow (P\{m \leftarrow m'\} \mid Q)\{m \leftarrow m'\} \models \mathcal{B} \\ &\Leftrightarrow \forall Q. Q \models \mathcal{A} \Rightarrow P\{m \leftarrow m'\} \mid Q \models \mathcal{B} \\ &\Leftrightarrow P\{m \leftarrow m'\} \models \mathcal{A} \triangleright \mathcal{B} \end{aligned}$$

Case $n[\mathcal{A}]$. We have $m, m' \notin \{n\} \cup \text{fn}(\mathcal{A})$. With appeal to Lemma 2 and Lemma 6, and the induction hypothesis, we calculate:

$$\begin{aligned}
P \models n[\mathcal{A}] &\Leftrightarrow \exists P'. P \equiv n[P'] \wedge P' \models \mathcal{A} \\
&\Leftrightarrow \exists P'. P \equiv n[P'] \wedge P'\{m \leftarrow m'\} \models \mathcal{A} \\
&\Leftrightarrow \exists P', P''. P \equiv n[P'] \wedge P'\{m \leftarrow m'\} \equiv P'' \wedge P'' \models \mathcal{A} \\
&\Leftrightarrow \exists P''. P\{m \leftarrow m'\} \equiv n[P''] \wedge P'' \models \mathcal{A} \\
&\Leftrightarrow P\{m \leftarrow m'\} \models n[\mathcal{A}]
\end{aligned}$$

Case $\mathcal{A}@n$. We have $m, m' \notin \{n\} \cup \text{fn}(\mathcal{A})$. With appeal to the induction hypothesis, we calculate:

$$\begin{aligned}
P \models \mathcal{A}@n &\Leftrightarrow n[P] \models \mathcal{A} \\
&\Leftrightarrow (n[P])\{m \leftarrow m'\} \models \mathcal{A} \\
&\Leftrightarrow n[P\{m \leftarrow m'\}] \models \mathcal{A} \\
&\Leftrightarrow P\{m \leftarrow m'\} \models \mathcal{A}@n
\end{aligned}$$

□

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) *Suppose $m \notin \text{fn}(\mathcal{A} \triangleright \mathcal{B})$. Then:*

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

Proof The forwards direction is immediate. For the backwards direction, assume that $(\forall Q \in \text{Tree}_{\text{fn}(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}}. Q \models \mathcal{A} \Rightarrow P \mid Q \models \mathcal{B})$ and consider any tree Q such that $Q \models \mathcal{A}$. Suppose that $\text{fn}(P \mid Q) \subseteq \text{fn}(\mathcal{A} \triangleright \mathcal{B}) \cup \{m, n_1, \dots, n_k\}$ where $\{n_1, \dots, n_k\} \cap (\text{fn}(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}) = \emptyset$. 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 $Q' \in \text{Tree}_{\text{fn}(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}}$ and $Q' \models \mathcal{A}$, we obtain $P \mid Q' \models \mathcal{B}$ by assumption. Now, we have:

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

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: $P \mid \dots \mid P$.*

Definition 2 (Size of Trees)

$|P|^{\text{hw}} \triangleq (h, w)$ iff there are $a_1, n_1, P_1, \dots, a_k, n_k, P_k$, for some k , such that:

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

When $|P|^{\text{hw}} = (h, w)$, we write $|P|^h$ for h and $|P|^w$ 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 $|P|^h$ is the height of P , and $|P|^w$ is the width, defined as the maximum *multiplicity* of the subtrees of P . The multiplicity is the number of structurally equivalent non-empty trees under the same edge. For example:

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

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

Size of Logical Formulas

$ \mathbf{F} ^h$	$\triangleq 0$	$ \mathbf{F} ^w$	$\triangleq 0$
$ \mathcal{A} \wedge \mathcal{B} ^h$	$\triangleq \max(\mathcal{A} ^h, \mathcal{B} ^h)$	$ \mathcal{A} \wedge \mathcal{B} ^w$	$\triangleq \max(\mathcal{A} ^w, \mathcal{B} ^w)$
$ \mathcal{A} \Rightarrow \mathcal{B} ^h$	$\triangleq \max(\mathcal{A} ^h, \mathcal{B} ^h)$	$ \mathcal{A} \Rightarrow \mathcal{B} ^w$	$\triangleq \max(\mathcal{A} ^w, \mathcal{B} ^w)$
$ \mathbf{0} ^h$	$\triangleq 1$	$ \mathbf{0} ^w$	$\triangleq 1$
$ \mathcal{A} \mid \mathcal{B} ^h$	$\triangleq \max(\mathcal{A} ^h, \mathcal{B} ^h)$	$ \mathcal{A} \mid \mathcal{B} ^w$	$\triangleq \mathcal{A} ^w + \mathcal{B} ^w$
$ \mathcal{A} \triangleright \mathcal{B} ^h$	$\triangleq \mathcal{B} ^h$	$ \mathcal{A} \triangleright \mathcal{B} ^w$	$\triangleq \mathcal{B} ^w$
$ n[\mathcal{A}] ^h$	$\triangleq 1 + \mathcal{A} ^h$	$ n[\mathcal{A}] ^w$	$\triangleq \max(2, \mathcal{A} ^w)$
$ \mathcal{A} @ n ^h$	$\triangleq \max(\mathcal{A} ^h - 1, 0)$	$ \mathcal{A} @ n ^w$	$\triangleq \mathcal{A} ^w$

Here are the sizes for the derived propositional connectives:

$ \mathbf{T} ^h$	$\triangleq 0$	$ \mathbf{T} ^w$	$\triangleq 0$
$ \neg \mathcal{A} ^h$	$\triangleq \mathcal{A} ^h$	$ \neg \mathcal{A} ^w$	$\triangleq \mathcal{A} ^w$
$ \mathcal{A} \vee \mathcal{B} ^h$	$\triangleq \max(\mathcal{A} ^h, \mathcal{B} ^h)$	$ \mathcal{A} \vee \mathcal{B} ^w$	$\triangleq \max(\mathcal{A} ^w, \mathcal{B} ^w)$

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 $P \sim_{h,w} Q$)

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

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 $P \sim_{h,w} R$.*

Proof By an easy induction on h . □

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

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

Proof By induction on h . The case $h = 0$ is immediate.

For $h + 1$, suppose $P \sim_{h+1,w} Q$ and $(h', w') \leq (h + 1, w)$. If $h' = 0$ then clearly $P \sim_{h',w'} Q$. If $h' = h'' + 1$ for some h'' , then consider any $i \in 1..w'$, n , P_j for $j \in 1..i$ such that

$$P \equiv n[P_1] \mid \cdots \mid n[P_i] \mid P_{i+1}$$

Since $w' \leq w$, then $i \in 1..w$, and from $P \sim_{h+1,w} Q$ we have

$$Q \equiv n[Q_1] \mid \cdots \mid n[Q_i] \mid Q_{i+1} \text{ such that } P_j \sim_{h,w} Q_j \text{ for } j \in 1..i$$

Since $(h'', w') \leq (h, w)$, by induction hypothesis we have $P_j \sim_{h'',w'} Q_j$ for $j \in 1..i$. This proves $P \sim_{h''+1,w'} Q$, that is, $P \sim_{h',w'} Q$. □

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 $P \mid Q \sim_{h,w} P' \mid Q'$.*

Proof We prove both parts directly.

- (1) Suppose $P \sim_{h,w} Q$. If $w = 0$ then the conclusion is immediate. Otherwise, consider any $i \in 1..w$, n, P_j for $j \in 1..i$ such that

$$n[P] \equiv n[P_1] \mid \cdots \mid n[P_i] \mid P_{i+1}$$

Then $i = 1$ and $P_1 \equiv P$ and $P_{i+1} \equiv \mathbf{0}$. We have $n[Q] \equiv n[Q] \mid \mathbf{0}$, and $P_1 \sim_{h,w} Q$ by Lemma 9. This proves $n[P] \sim_{h+1,w} n[Q]$.

- (2) There are two cases. If $h = 0$ then the conclusion is immediate.

For $h + 1$, suppose $P \sim_{h+1,w} P'$ and $Q \sim_{h+1,w} Q'$; then consider any $i \in 1..w$, n, R_j for $j \in 1..i$ such that

$$P \mid Q \equiv n[R_1] \mid \cdots \mid n[R_i] \mid R_{i+1}$$

Suppose without loss of generality that the R_j are ordered in a way that there exist $k \in 1..i, P_{\dagger}, Q_{\dagger}$ such that

$$\begin{aligned} P &\equiv n[R_1] \mid \cdots \mid n[R_k] \mid P_{\dagger} \\ Q &\equiv n[R_{k+1}] \mid \cdots \mid n[R_i] \mid Q_{\dagger} \\ R_{i+1} &\equiv P_{\dagger} \mid Q_{\dagger} \end{aligned}$$

Since $k \in 1..w$, from $P \sim_{h+1,w} P'$ we have

$$P' \equiv n[P'_1] \mid \cdots \mid n[P'_k] \mid P'_{\dagger} \text{ such that } R_j \sim_{h,w} P'_j \text{ for } j \in 1..k$$

Similarly, from $Q \sim_{h+1,w} Q'$ we have

$$Q' \equiv n[Q'_{k+1}] \mid \cdots \mid n[Q'_i] \mid Q'_{\dagger} \text{ such that } R_j \sim_{h,w} Q'_j \text{ for } j \in (k+1)..i$$

Hence, we have

$$P' \mid Q' \equiv n[P'_1] \mid \cdots \mid n[P'_k] \mid n[Q'_{k+1}] \mid \cdots \mid n[Q'_i] \mid P'_{\dagger} \mid Q'_{\dagger}$$

Since $R_j \sim_{h,w} P'_j$ for $j \in 1..k$ and $R_j \sim_{h,w} Q'_j$ for $j \in (k+1)..i$, this proves $P \mid Q \sim_{h+1,w} P' \mid Q'$. \square

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 $P'' \sim_{h,w_2} Q''$.*

Proof There are two cases. If $h = 0$ then the conclusion is immediate.

For $h + 1$, suppose $P' \mid P'' \sim_{h+1,w_1+w_2} Q$. Consider the following definition:

A tree P is in (h, w) -normal form if whenever $P \equiv n[P_1] \mid n[P_2] \mid P_3$ for some P_1, P_2, P_3 , if $P_1 \sim_{h,w} P_2$ then $P_1 \equiv P_2$. Note that $P \sim_{h+1,w} n[P_1] \mid n[P_1] \mid P_3$, hence it is always possible to find a P^{\dagger} such that P^{\dagger} is in (h, w) -normal form and $P \sim_{h+1,w} P^{\dagger}$.

We can assume without loss of generality that P' and P'' are in (h, w) -normal form, and by Lemma 10 that Q is in (h, w) -normal form. Hence, there exist k, P_j, a'_j, a''_j, b_j for $j \in 1..k$ such that

$$\begin{aligned} P' &\equiv a'_1 \cdot n_1[P_1] \mid \cdots \mid a'_k \cdot n_k[P_k] \\ P'' &\equiv a''_1 \cdot n_1[P_1] \mid \cdots \mid a''_k \cdot n_k[P_k] \\ Q &\equiv b_1 \cdot n_1[P_1] \mid \cdots \mid b_k \cdot n_k[P_k] \end{aligned}$$

where if $P_i \sim_{h,w} P_j$ and $n_i = n_j$ then $i = j$.

To split Q into two parts, we now specify how to split each b_i , for $i \in 1..k$, into b'_i and b''_i , such that:

$$\begin{aligned} b_i &= b'_i + b''_i \\ a'_i \cdot n_i[P_i] &\sim_{h+1,w_1} b'_i \cdot n_i[P_i] \\ a''_i \cdot n_i[P_i] &\sim_{h+1,w_2} b''_i \cdot n_i[P_i] \end{aligned}$$

For each $i \in 1..k$, we choose b'_i and b''_i according to the following cases:

- Case $a'_i + a''_i < w_1 + w_2$. Then $P' \mid P'' \sim_{h+1,w} Q$ implies $b_i = a'_i + a''_i$, so we can choose $b'_i = a'_i$ and $b''_i = a''_i$.
- Case $a'_i + a''_i \geq w_1 + w_2$. Then $P' \mid P'' \sim_{h+1,w} Q$ implies $b_i \geq w_1 + w_2$. There are three subcases:
 - Subcase $a'_i \geq w_1$ and $a''_i \geq w_2$. Then we choose $b'_i = w_1$ and $b''_i = b_i - w_1$ (note that b''_i is saturated, that is, $b''_i \geq w_2$, since $b_i \geq w_1 + w_2$).
 - Subcase $a'_i < w_1$. We must have $a''_i \geq w_2$. Then we choose $b'_i = a'_i$ and $b''_i = b_i - a'_i$. So $b''_i \geq w_2$ since $b_i \geq w_1 + w_2$ and $b'_i < w_1$.
 - Subcase $a''_i < w_2$. This is symmetric to the previous case. We must have $a'_i \geq w_1$. We choose $b''_i = a''_i$ and $b'_i = b_i - a''_i$. So $b'_i \geq w_1$ since $b_i \geq w_1 + w_2$ and $b''_i < w_2$.

Now we define Q' and Q'' as follows:

$$Q' \equiv b'_1 \cdot n_1[P_1] \mid \cdots \mid b'_k \cdot n_k[P_k] \quad Q'' \equiv b''_1 \cdot n_1[P_1] \mid \cdots \mid b''_k \cdot n_k[P_k]$$

We have $Q \equiv Q' \mid Q''$, and by repeated application of Lemma 11 we get $P' \sim_{h+1,w_1} Q'$ and $P'' \sim_{h+1,w_2} Q''$. \square

Proposition 2 *If $|\mathcal{A}|^{\text{hw}} = (h, w)$ 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} . We consider only some interesting cases.

Case 0. Suppose $P \models \mathbf{0}$ and $P \sim_{1,1} Q$. Then $P \equiv \mathbf{0}$. Since $P \sim_{1,1} Q$, if $Q \equiv n[Q_1] \mid Q_2$ for some n, Q_1, Q_2 then $P \equiv n[P_1] \mid P_2$ for some P_1, P_2 . Hence $Q \equiv \mathbf{0}$; thus $Q \models \mathbf{0}$.

Case $\mathcal{A}_1 \mid \mathcal{A}_2$. Suppose $|\mathcal{A}_i|^{\text{hw}} = (h_i, w_i)$ for $i = 1, 2$ and $P \models \mathcal{A}_1 \mid \mathcal{A}_2$. We have $|(\mathcal{A}_1 \mid \mathcal{A}_2)|^{\text{hw}} = (\max(h_1, h_2), w_1 + w_2)$ and there exist P_1, P_2 such that $P \equiv P_1 \mid P_2$ and $P_i \models \mathcal{A}_i$ for $i = 1, 2$. Then by Lemma 12 there exist Q_1, Q_2 such that $Q \equiv Q_1 \mid Q_2$ and $P_i \sim_{\max(h_1, h_2), w_i} Q_i$ for $i = 1, 2$. Then $P_i \sim_{h_i, w_i} Q_i$ for $i = 1, 2$ by Lemma 10, hence $Q_i \models \mathcal{A}_i$ for $i = 1, 2$ by induction hypothesis. This proves $Q \models \mathcal{A}_1 \mid \mathcal{A}_2$.

Case $\mathcal{A} \triangleright \mathcal{B}$. Suppose $|\mathcal{B}|^{\text{hw}} = (h, w)$ and $P \models \mathcal{A} \triangleright \mathcal{B}$. We have $|\mathcal{A} \triangleright \mathcal{B}|^{\text{hw}} = (h, w)$ and $P \sim_{h, w} Q$. Consider any P_1 such that $P_1 \models \mathcal{A}$; then $P \mid P_1 \models \mathcal{B}$. Since $P \sim_{h, w} Q$ and $P_1 \sim_{h, w} P_1$ we have $P \mid P_1 \sim_{h, w} Q \mid P_1$ by Lemma 11. Hence $Q \mid P_1 \models \mathcal{B}$ by induction hypothesis. This proves $Q \models \mathcal{A} \triangleright \mathcal{B}$.

Case $n[\mathcal{A}]$. Suppose $|\mathcal{A}|^{\text{hw}} = (h, w)$. We have $|n[\mathcal{A}]|^{\text{hw}} = (h + 1, \max(w, 2))$ and $P \sim_{h+1, \max(w, 2)} Q$ and $P \models n[\mathcal{A}]$. Then there exists P' such that $P \equiv n[P']$ and $P' \models \mathcal{A}$. From $P \sim_{h+1, \max(w, 2)} Q$ we deduce that there exists Q' such that $Q \equiv n[Q']$ and $P' \sim_{h, \max(w, 2)} Q'$. Lemma 10 implies $P' \sim_{h, w} Q'$, and by induction hypothesis we have $Q' \models \mathcal{A}$. This proves $Q \models n[\mathcal{A}]$.

Case $\mathcal{A} @ n$. Suppose $|\mathcal{A}|^{\text{hw}} = (h, w)$. We have $|\mathcal{A} @ n|^{\text{hw}} = (\max(h - 1, 0), w)$ and $P \sim_{\max(h-1, 0), w} Q$. If $h > 0$ then we have $n[P] \sim_{h, w} n[Q]$ by Lemma 11. If $h = 0$ then $n[P] \sim_{h, w} n[Q]$ is immediate. With appeal to the induction hypothesis, we calculate:

$$\begin{aligned} P \models \mathcal{A} @ n &\Leftrightarrow n[P] \models \mathcal{A} \\ &\Leftrightarrow n[Q] \models \mathcal{A} \\ &\Leftrightarrow Q \models \mathcal{A} @ n \end{aligned}$$

□

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 $P \in \text{Tree}_X, h, w$ there exists $P' \in \text{Tree}_X$ such that $P \sim_{h, w} P'$ and $|P'|^{\text{hw}} \leq (h, w)$.*

Proof We describe how to construct P' by induction on h . For $h = 0$ define $P' \triangleq \mathbf{0}$.

For $h + 1$, suppose $P \equiv n_1[P_1] \mid \cdots \mid n_k[P_k]$, for some k and n_j, P_j with $j \in 1..k$. Let P'_j , for $j \in 1..k$, be the tree obtained by pruning P_j to size h, w . Define $Q \triangleq n_1[P'_1] \mid \cdots \mid n_k[P'_k]$. We can write Q in a canonical form with respect to \equiv , that is, there exist i and a_j, m_j, Q_j for $j \in 1..i$ such that $Q \equiv a_1 \cdot m_1[Q_1] \mid \cdots \mid a_i \cdot m_i[Q_i]$ and, for all $j, j' \in 1..i$, if $m_j[Q_j] \equiv m_{j'}[Q_{j'}]$ then $j = j'$. For each $j \in 1..i$, define $b_i \triangleq \min(a_i, w)$. Then we can define $P' \triangleq b_1 \cdot m_1[Q_1] \mid \cdots \mid b_i \cdot m_i[Q_i]$. It is easy to see that $|P'|^{\text{hw}} \leq (h + 1, w)$ and $P \sim_{h+1, w} P'$. □

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

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

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

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

□

Proposition 4 (Bounding Size and Names) For any tree P and formulas \mathcal{A} and \mathcal{B} , if $m \notin \text{fn}(\mathcal{A} \triangleright \mathcal{B})$ and $X = \text{fn}(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}$ and $h = \max(|\mathcal{A}|^h, |\mathcal{B}|^h)$ and $w = \max(|\mathcal{A}|^w, |\mathcal{B}|^w)$, then:

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

Proof We have:

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

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 $\text{fn}(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}$, and whose size is bounded by $\max(|\mathcal{A}|^{\text{hw}}, |\mathcal{B}|^{\text{hw}})$. 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:

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

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

Definition 5 If $X = \{n_1, \dots, n_k\}$, define $EQ_{h,w}^X$ as follows:

$$\begin{aligned}
EQ_{0,w}^X &\triangleq \{\langle 0 \rangle_{\equiv}\} \\
EQ_{h+1,w}^X &\triangleq (n_1[EQ_{h,w}^X] + \dots + n_k[EQ_{h,w}^X])^{\leq w}
\end{aligned}$$

Lemma 14 If $|P|^{\text{hw}} \leq (h, w)$ and $|P'|^{\text{hw}} \leq (h, w)$, then

- (1) $P \in \text{Tree}_X$ implies $\langle P \rangle_{\equiv} \in EQ_{h,w}^X$.
- (2) $P \equiv P' \iff P \sim_{h,w} P'$.

Proof Part (1) is a simple induction on h .

For Part (2), the interesting direction is \Leftarrow . We proceed by induction on h . If $h = 0$ then $|P|^{\text{h}} = |Q|^{\text{h}} = 0$, hence $P \equiv Q \equiv \mathbf{0}$.

For the case $h + 1$, suppose $|P|^{\text{hw}} \leq (h + 1, w)$ and $|P'|^{\text{hw}} \leq (h + 1, w)$ and $P \sim_{h+1,w} P'$. Write P and P' in canonical form with respect to \equiv , that is, there exist k and a_j, a'_j, n_j, P_j for $j \in 1..k$ such that

$$P \equiv a_1 \cdot n_1[P_1] \mid \dots \mid a_k \cdot n_k[P_k] \quad P' \equiv a'_1 \cdot n_1[P_1] \mid \dots \mid a'_k \cdot n_k[P_k]$$

where, for all $i, j \in 1..k$, if $n_i[P_i] \equiv n_j[P_j]$ then $i = j$. Since $|P|^{\text{hw}} \leq (h + 1, w)$ and $|P'|^{\text{hw}} \leq (h + 1, w)$ we have $a_j \leq w$ and $a'_j \leq w$ for each $j \in 1..k$. For each $i \in 1..k$ we show $a_i \leq a'_i$:

There exists P_{\dagger} such that $P \equiv a_i \cdot n_i[P_i] \mid P_{\dagger}$. Then by definition of $P \sim_{h+1,w} P'$ there exist $P'_1, \dots, P'_{a_i}, P'_{\dagger}$ such that $P' \equiv n_i[P'_1] \mid \dots \mid n_i[P'_{a_i}] \mid P'_{\dagger}$ and $P_i \sim_{h,w} P'_j$ for $j \in 1..a_i$. By induction hypothesis we have $P_i \equiv P'_j$ for each $j \in 1..a_i$, hence $P' \equiv a_i \cdot n_i[P_i] \mid P'_{\dagger}$. This proves $a_i \leq a'_i$.

With a symmetric argument we can show $a'_i \leq a_i$ for each $i \in 1..k$. This proves $P \equiv P'$. \square

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

Lemma 15 $\langle P \rangle_{\equiv} \in EQ_{h,w}^X \Leftrightarrow P \in \text{Tree}_X \wedge |P|^{\text{hw}} \leq (h, w)$.

Proof By construction, if $\langle P \rangle_{\equiv} \in EQ_{h,w}^X$ then $P \in \text{Tree}_X$ and $|P|^{\text{hw}} \leq (h, w)$. The converse follows from Lemma 14. \square

The following proposition shows that $EQ_{h,w}^X$ is an enumeration of the representatives of the equivalence classes in $\text{Tree}_X / \sim_{h,w}$.

Proposition 5 *The function $f : \text{Tree}_X \rightarrow \text{Tree}_X / \sim_{h,w}$ sending P to $\langle P \rangle_{h,w}$ extends to a bijection $f' : EQ_{h,w}^X \rightarrow \text{Tree}_X / \sim_{h,w}$.*

Proof Let f' be the function sending $\langle P \rangle_{\equiv}$ to $\langle P \rangle_{h,w}$. Clearly f' is well defined since $P \equiv P'$ implies $P \sim_{h,w} P'$.

To show that f' is surjective, take any $\langle P \rangle_{h,w} \in \text{Tree}_X / \sim_{h,w}$. By lemma 13 there exists $P' \in \text{Tree}_X$ such that $P \sim_{h,w} P'$ and $|P'|^{\text{hw}} \leq (h, w)$. So $\langle P' \rangle_{h,w} = \langle P \rangle_{h,w}$ and $\langle P' \rangle_{\equiv} \in EQ_{h,w}^X$ by lemma 14.

To show that f' is injective, consider any $P, Q \in \text{Tree}_X$ with $\langle P \rangle_{\equiv}, \langle Q \rangle_{\equiv} \in EQ_{h,w}^X$ and $\langle P \rangle_{h,w} = \langle Q \rangle_{h,w}$. Then $|P|^{\text{hw}} \leq (h, w)$ and $|Q|^{\text{hw}} \leq (h, w)$ by Lemma 15, hence $P \equiv Q$ by Lemma 14. This proves $\langle P \rangle_{\equiv} = \langle Q \rangle_{\equiv}$. \square

Theorem 1 (Finite Bound) *Consider any formulas \mathcal{A} and \mathcal{B} . Let $EQ_{h,w}^X = \{\langle Q_1 \rangle_{\equiv}, \dots, \langle Q_n \rangle_{\equiv}\}$, where $h = \max(|\mathcal{A}|^h, |\mathcal{B}|^h)$ and $w = \max(|\mathcal{A}|^w, |\mathcal{B}|^w)$ and $X = \text{fn}(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}$ for some $m \notin \text{fn}(\mathcal{A} \triangleright \mathcal{B})$.*

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:

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

\square

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 non-empty set $\{\langle Q, R \rangle \mid P \equiv Q \mid R\} / (\equiv \times \equiv)$.
 $T(\mathcal{A} \triangleright \mathcal{B})$ is the finite non-empty set $EQ_{h,w}^X$, where $h = \max(|\mathcal{A}|^h, |\mathcal{B}|^h)$ and $w = \max(|\mathcal{A}|^w, |\mathcal{B}|^w)$ and $X = \text{fn}(\mathcal{A} \triangleright \mathcal{B}) \cup \{m\}$ for some $m \notin \text{fn}(\mathcal{A} \triangleright \mathcal{B})$.
 $T(n, P)$ is the finite, possibly empty, 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} , $P \models \mathcal{A} \triangleright \mathcal{B} \Leftrightarrow \forall Q \in T(\mathcal{A} \triangleright \mathcal{B}). Q \models \mathcal{A} \Rightarrow Q \models \mathcal{B}$.
- (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 [CG00]. \square

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

Proof As noted in Section 2, Lemmas 4 and 5 establish the equivalence of satisfaction and validity. An algorithm for satisfaction follows from the rules of its definition in Section 2, together with the facts in Lemma 16. \square

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} . Suppose $EQ_{h,w}^X = \{\langle P_1 \rangle_{\equiv}, \dots, \langle P_n \rangle_{\equiv}\}$, where $(h, w) = |\mathcal{A}|^{hw}$ 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 $EQ_{h,w}^X$ 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 [CC02]. 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 (\mid L), (\triangleright R), and (n L).

Rules of the Sequent Calculus:

$\frac{(\text{Id}) \quad P \equiv Q}{\Gamma, P : \mathcal{A} \vdash Q : \mathcal{A}, \Delta}$	$\frac{(\text{Cut}) \quad \Gamma \vdash P : \mathcal{A}, \Delta \quad \Gamma, P : \mathcal{A} \vdash \Delta}{\Gamma \vdash \Delta}$
$\frac{(\text{C L}) \quad \Gamma, P : \mathcal{A}, P : \mathcal{A} \vdash \Delta}{\Gamma, P : \mathcal{A} \vdash \Delta}$	$\frac{(\text{C R}) \quad \Gamma \vdash P : \mathcal{A}, P : \mathcal{A}, \Delta}{\Gamma \vdash P : \mathcal{A}, \Delta}$
$\frac{(\mathbf{F L})}{\Gamma, P : \mathbf{F} \vdash \Delta}$	$\frac{(\mathbf{F R}) \quad \Gamma \vdash \Delta}{\Gamma \vdash P : \mathbf{F}, \Delta}$
$\frac{(\wedge \text{L}) \quad \Gamma, P : \mathcal{A}, P : \mathcal{B} \vdash \Delta}{\Gamma, P : \mathcal{A} \wedge \mathcal{B} \vdash \Delta}$	$\frac{(\wedge \text{R}) \quad \Gamma \vdash P : \mathcal{A}, \Delta \quad \Gamma \vdash P : \mathcal{B}, \Delta}{\Gamma \vdash P : \mathcal{A} \wedge \mathcal{B}, \Delta}$

$$\begin{array}{c}
\begin{array}{c}
(\Rightarrow \text{L}) \\
\frac{\Gamma \vdash P : \mathcal{A}, \Delta \quad \Gamma, P : \mathcal{B} \vdash \Delta}{\Gamma, P : \mathcal{A} \Rightarrow \mathcal{B} \vdash \Delta}
\end{array}
\qquad
\begin{array}{c}
(\Rightarrow \text{R}) \\
\frac{\Gamma, P : \mathcal{A} \vdash P : \mathcal{B}, \Delta}{\Gamma \vdash P : \mathcal{A} \Rightarrow \mathcal{B}, \Delta}
\end{array} \\
\\
\begin{array}{c}
(\mathbf{0} \text{L}) \\
\frac{P \neq \mathbf{0}}{\Gamma, P : \mathbf{0} \vdash \Delta}
\end{array}
\qquad
\begin{array}{c}
(\mathbf{0} \text{R}) \\
\frac{P \equiv \mathbf{0}}{\Gamma \vdash P : \mathbf{0}, \Delta}
\end{array} \\
\\
\begin{array}{c}
(| \text{L}) \\
\frac{\forall \langle Q, R \rangle \in T(P). \Gamma, Q : \mathcal{A}, R : \mathcal{B} \vdash \Delta}{\Gamma, P : \mathcal{A} | \mathcal{B} \vdash \Delta}
\end{array} \\
\\
\begin{array}{c}
(| \text{R}) \\
\frac{\Gamma \vdash Q : \mathcal{A}, \Delta \quad \Gamma \vdash R : \mathcal{B}, \Delta \quad P \equiv Q | R}{\Gamma \vdash P : \mathcal{A} | \mathcal{B}, \Delta}
\end{array} \\
\\
\begin{array}{c}
(\triangleright \text{L}) \\
\frac{\Gamma \vdash Q : \mathcal{A}, \Delta \quad \Gamma, Q | P : \mathcal{B} \vdash \Delta}{\Gamma, P : \mathcal{A} \triangleright \mathcal{B} \vdash \Delta}
\end{array}
\qquad
\begin{array}{c}
(\triangleright \text{R}) \\
\frac{\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}
\end{array} \\
\\
\begin{array}{c}
(n[] \text{L}) \\
\frac{\forall Q \in T(n, P). \Gamma, Q : \mathcal{A} \vdash \Delta}{\Gamma, P : n[\mathcal{A}] \vdash \Delta}
\end{array}
\qquad
\begin{array}{c}
(n[] \text{R}) \\
\frac{\Gamma \vdash Q : \mathcal{A}, \Delta \quad P \equiv n[Q]}{\Gamma \vdash P : n[\mathcal{A}], \Delta}
\end{array} \\
\\
\begin{array}{c}
(@n \text{L}) \\
\frac{\Gamma, n[P] : \mathcal{A} \vdash \Delta}{\Gamma, P : \mathcal{A}@n \vdash \Delta}
\end{array}
\qquad
\begin{array}{c}
(@n \text{R}) \\
\frac{\Gamma \vdash n[P] : \mathcal{A}, \Delta}{\Gamma \vdash P : \mathcal{A}@n, \Delta}
\end{array}
\end{array}$$

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 [CC02]. 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).*

Proof By induction on the derivation of $\Gamma \vdash \Delta$. The second part of the statement comes from inspection of the cases different from (Id), (Cut), (C L), (C R). \square

4.2 Soundness and Completeness

We make a conventional interpretation of sequents:

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

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 $[\Gamma \vdash n[P] : \mathcal{A}', \Delta]$

Proof By detailed, but straightforward, calculations. □

Lemma 19 (Finite Test Sets)

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

Proof By expanding definitions, and appeal to Lemma 16. \square

Theorem 3 (Soundness) If $\Gamma \vdash \Delta$ is derivable, $\llbracket \Gamma \vdash \Delta \rrbracket$.

Proof By induction on the derivation of $\Gamma \vdash \Delta$. \square

Theorem 4 (Completeness) If $\llbracket \Gamma \vdash \Delta \rrbracket$, 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 (\mid L), ($n\llbracket$ L) and, particularly, (\triangleright R), relying on Lemma 19. These are the only cases we show.

Subcase $\llbracket \Gamma, P : \mathcal{A}' \mid \mathcal{A}'' \vdash \Delta \rrbracket$. By Lemma 18(\mid L) we have $\forall P', P''. P \equiv P' \mid P'' \Rightarrow \llbracket \Gamma, P' : \mathcal{A}', P'' : \mathcal{A}'' \vdash \Delta \rrbracket$. By Lemma 19(1) there is a finite set $T(P)$ such that $\forall \langle P', P'' \rangle \in T(P). \llbracket \Gamma, P' : \mathcal{A}', P'' : \mathcal{A}'' \vdash \Delta \rrbracket$. By IndHyp, $\forall \langle P', P'' \rangle \in T(P). \Gamma, P' : \mathcal{A}', P'' : \mathcal{A}'' \vdash \Delta$ has a derivation. Hence by (\mid L) we can construct a (finite) derivation for $\Gamma, P : \mathcal{A}' \mid \mathcal{A}'' \vdash \Delta$.

Subcase $\llbracket \Gamma, P : n[\mathcal{A}'] \vdash \Delta \rrbracket$. By Lemma 18($n\llbracket$ L) we have $\forall P'. P \equiv n[P'] \Rightarrow \llbracket \Gamma, P' : \mathcal{A}' \vdash \Delta \rrbracket$. By Lemma 19(3) there is a finite set $T(n, P)$ such that $\forall P' \in T(n, P). \llbracket \Gamma, P' : \mathcal{A}' \vdash \Delta \rrbracket$. By IndHyp, $\forall P' \in T(n, P). \Gamma, P' : \mathcal{A}' \vdash \Delta$ has a derivation. Hence by ($n\llbracket$ L) we can construct a (finite) derivation for $\Gamma, P : n[\mathcal{A}'] \vdash \Delta$.

Subcase $\llbracket \Gamma \vdash P : \mathcal{A}' \triangleright \mathcal{A}'', \Delta \rrbracket$. By Lemma 18(\triangleright R) we have $\forall P'. \llbracket \Gamma, P' : \mathcal{A}' \vdash P' \mid P : \mathcal{A}'', \Delta \rrbracket$. By Lemma 19(2) there is a finite set $T(\mathcal{A}' \triangleright \mathcal{A}'')$ such that $\forall P' \in T(\mathcal{A}' \triangleright \mathcal{A}''). \llbracket \Gamma, P' : \mathcal{A}' \vdash P' \mid P : \mathcal{A}'', \Delta \rrbracket$. By IndHyp, $\forall P' \in T(\mathcal{A}' \triangleright \mathcal{A}''). \Gamma, P' : \mathcal{A}' \vdash P' \mid P : \mathcal{A}'', \Delta$ has a derivation. Hence by (\triangleright R) we can construct a (finite) derivation for $\Gamma \vdash P : \mathcal{A}' \triangleright \mathcal{A}'', \Delta$.

For the second part of the statement, it is sufficient to note that the rules (Id), (Cut), (C L), (C R) are never used in the proof to construct the derivation, and that the cases ($\mathbf{0}$ L), ($\mathbf{0}$ R), ($n\llbracket$ R) use Lemma 17 applied to a derivation that, inductively, does not contain (Id), (Cut), (C L), (C R). \square

Proposition 6 (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 $\llbracket \Gamma \vdash \Delta \rrbracket$ 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). \square

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

Proposition 7 (Decidability) *It is decidable whether $\Gamma \vdash \Delta$ is derivable.*

Proof Suppose that $\Gamma = P_1 : \mathcal{A}_1, \dots, P_n : \mathcal{A}_n$ and $\Delta = Q_1 : \mathcal{B}_1, \dots, Q_m : \mathcal{B}_m$. By Theorems 3 (Soundness) and 4 (Completeness), $P_1 : \mathcal{A}_1, \dots, P_n : \mathcal{A}_n \vdash Q_1 : \mathcal{B}_1, \dots, Q_m : \mathcal{B}_m$ is derivable if and only if $\wedge \llbracket P_1 : \mathcal{A}_1, \dots, P_n : \mathcal{A}_n \rrbracket \Rightarrow \vee \llbracket Q_1 : \mathcal{B}_1, \dots, Q_m : \mathcal{B}_m \rrbracket$. By Theorem 2 we know that $P \models \mathcal{A}$ is decidable. Therefore, we just need to test that either there is an i with $P_i \not\models \mathcal{A}_i$, or there is a j with $Q_j \models \mathcal{B}_j$. \square

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), (\mid R) and ($n \llbracket$ 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 7.

Lemma 20 (More on Finite Test Sets)

- (1) *For any P there is a finite set $T(P)$ with:*
 $\exists P', P''. P \equiv P' \mid P'' \wedge \llbracket \Gamma \vdash P' : \mathcal{A}', \Delta \rrbracket \wedge \llbracket \Gamma \vdash P'' : \mathcal{A}'', \Delta \rrbracket$
iff $\exists (P', P'') \in T(P). \llbracket \Gamma \vdash P' : \mathcal{A}', \Delta \rrbracket \wedge \llbracket \Gamma \vdash P'' : \mathcal{A}'', \Delta \rrbracket$.
- (2) *For any $\mathcal{A}', \mathcal{A}''$, there is a finite set $T(\mathcal{A}' \triangleright \mathcal{A}'')$ with:*
 $\exists P'. \llbracket \Gamma \vdash P' : \mathcal{A}', \Delta \rrbracket \wedge \llbracket \Gamma, P' \mid P : \mathcal{A}'' \vdash \Delta \rrbracket$
iff $\exists P' \in T(\mathcal{A}' \triangleright \mathcal{A}''). \llbracket \Gamma \vdash P' : \mathcal{A}', \Delta \rrbracket \wedge \llbracket \Gamma, P' \mid P : \mathcal{A}'' \vdash \Delta \rrbracket$.
- (3) *For any P there is a finite set $T(n, P)$ with:*
 $\exists P'. P \equiv n[P'] \wedge \llbracket \Gamma \vdash P' : \mathcal{A}', \Delta \rrbracket$
iff $\exists P' \in T(n, P). \llbracket \Gamma \vdash P' : \mathcal{A}', \Delta \rrbracket$.

Proof By expanding definitions, and appeal to Lemma 16. \square

Theorem 5 (Complete Proof Procedure) *For any $\Gamma \vdash \Delta$ there is a procedure such that: if $\neg \llbracket \Gamma \vdash \Delta \rrbracket$, then the procedure terminates with failure; if $\llbracket \Gamma \vdash \Delta \rrbracket$, 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 $size \geq 1$, 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 $P \neq \mathbf{0}$ (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 $size \geq 1$, 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 $\Gamma \vdash P : n[\mathcal{A}'], \Delta$.

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. \square

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:

$$\begin{aligned} \mathbf{T} &\rightarrow \neg\mathbf{0} \\ ((\mathcal{A} \wedge \neg\mathbf{0}) \mid n[\mathcal{B}]) &\rightarrow (n[\mathcal{A}] \mid \mathcal{B}) \end{aligned}$$

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 $\mathbf{vld}(\mathcal{A} \Rightarrow \mathcal{B})$. 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)
$\mathcal{F} \rightarrow \mathcal{G}$	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

<i>case t of</i> $\mathbf{0}.u_1,$ $n[x:\mathcal{A}].u_2,$ $(x:\mathcal{A} \mid y:\mathcal{B}).u_3,$ <i>else</i> u_4 \triangleq $t?(z_1:\mathbf{0}).u_1,$ $t?(z_2:n[\mathcal{A}]).z_2?n[x:\mathcal{A}].u_2,$ $t?(z_3:\mathcal{A} \mid \mathcal{B}).z_3?(x:\mathcal{A} \mid y:\mathcal{B}).u_3,$ u_4	<i>analyze t</i> if $t \equiv \mathbf{0}$, run u_1 , else if $t \equiv n[P]$ and $P \models A$, bind P to x and run u_2 , else 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 run u_4 can be translated as:
--	--

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

$\frac{}{\mathbf{0} \Downarrow_{\rho} \mathbf{0}}$	$\frac{}{t \mid u \Downarrow_{\rho} P \mid Q}$	$\frac{}{n[t] \Downarrow_{\rho} n[P]}$
$\frac{}{t \Downarrow_{\rho} \equiv n[P]} \quad P \models \mathcal{A} \quad u \Downarrow_{\rho[x \leftarrow P]} F$	$\frac{}{t?(x:\mathcal{A}).u \Downarrow_{\rho} F}$	
	$\frac{}{t \Downarrow_{\rho} \equiv P' \mid P''} \quad P' \models \mathcal{A} \quad P'' \models \mathcal{B}$	$\frac{}{u \Downarrow_{\rho[x \leftarrow P'][y \leftarrow P'']} F}$
$\frac{}{t?(x:\mathcal{A} \mid y:\mathcal{B}).u \Downarrow_{\rho} F}$		
$\frac{}{t \Downarrow_{\rho} P} \quad P \models \mathcal{A} \quad u \Downarrow_{\rho[x \leftarrow P]} F$	$\frac{}{t?(x:\mathcal{A}).u, v \Downarrow_{\rho} F}$	
	$\frac{}{t \Downarrow_{\rho} P} \quad P \models \neg \mathcal{A} \quad v \Downarrow_{\rho[x \leftarrow P]} F$	$\frac{}{t?(x:\mathcal{A}).u, v \Downarrow_{\rho} F}$
$\frac{}{x \in \text{dom}(\rho)}$	$\frac{}{\lambda x:\mathcal{F}.t \Downarrow_{\rho} \langle \rho, x, t \rangle}$	
$\frac{}{t \Downarrow_{\rho} \langle \rho', x, t' \rangle} \quad u \Downarrow_{\rho} G \quad t' \Downarrow_{\rho'[x \leftarrow G]} H$		
$\frac{}{t(u) \Downarrow_{\rho} H}$		

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 $\text{dom}(E)$ 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 $x \notin \text{dom}(E)$), and by $E(x)$ the type associated with x in E (provided that $x \in \text{dom}(E)$).

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:

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

(Term 0) $\frac{E \vdash \diamond}{E \vdash \mathbf{0} : \mathbf{0}}$	(Term) $\frac{E \vdash t : \mathcal{A} \quad E \vdash u : \mathcal{B}}{E \vdash t \mid u : \mathcal{A} \mid \mathcal{B}}$	(Term n[]) $\frac{E \vdash t : \mathcal{A}}{E \vdash n[t] : n[\mathcal{A}]}$
(Term ?) $\frac{E \vdash t : \mathcal{A} \mid \mathcal{B} \quad E, x:\mathcal{A}, y:\mathcal{B} \vdash u : \mathcal{F}}{E \vdash t?(x:\mathcal{A} \mid y:\mathcal{B}).u : \mathcal{F}}$	(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) $\frac{E \vdash \diamond}{E \vdash x : E(x)}$	(Term Lam) $\frac{E, x:\mathcal{F} \vdash t : \mathcal{G}}{E \vdash \lambda x:\mathcal{F}.t : \mathcal{F} \rightarrow \mathcal{G}}$	(Term App) $\frac{E \vdash t : \mathcal{F} \rightarrow \mathcal{G} \quad E \vdash u : \mathcal{F}}{E \vdash t(u) : \mathcal{G}}$
(Subsumption) $\frac{E \vdash t : \mathcal{F} \quad \mathcal{F} <: \mathcal{G}}{E \vdash t : \mathcal{G}}$	(Sub Tree) $\frac{\mathbf{vld}(\mathcal{A} \Rightarrow \mathcal{B})}{A <: B}$	(Sub \rightarrow) $\frac{\mathcal{F}' <: \mathcal{F} \quad \mathcal{G} <: \mathcal{G}'}{\mathcal{F} \rightarrow \mathcal{G} <: \mathcal{F}' \rightarrow \mathcal{G}'}$

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 $\langle \rho, x, t \rangle \models \mathcal{F} \rightarrow \mathcal{G}$ 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
$H \models \mathcal{F} \rightarrow \mathcal{G}$ 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 $\forall x \in \text{dom}(E). \rho(x) \models E(x)$	

Proposition 8 (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 \mathbf{vld} .

(Sub \rightarrow) We have $\mathcal{F}' <: \mathcal{F}$ and $\mathcal{G} <: \mathcal{G}'$, and $H \models \mathcal{F} \rightarrow \mathcal{G}$. 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 $\langle \rho, x, t \rangle \models \mathcal{F}' \rightarrow \mathcal{G}'$. \square

Proposition 9 (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 $t?(x:\mathcal{A} \mid y:\mathcal{B}).u \Downarrow_\rho F$. 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 $\rho[x \leftarrow P'] [y \leftarrow P''] \models E, x:\mathcal{A}, y:\mathcal{B}$. By Ind Hyp $E, x:\mathcal{A}, y:\mathcal{B} \vdash u : \mathcal{F}$ and $\rho[x \leftarrow P'] [y \leftarrow P''] \models E, x:\mathcal{A}, y:\mathcal{B}$ 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 $E \vdash \lambda x:\mathcal{F}.t : \mathcal{F} \rightarrow \mathcal{G}$ 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 $\langle \rho, x, t \rangle \models \mathcal{F} \rightarrow \mathcal{G}$, 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 $E \vdash t : \mathcal{G} \rightarrow \mathcal{F}$ and $E \vdash u : \mathcal{G}$. We must have from (Red App) that $t \Downarrow_{\rho} \langle \rho', x, t' \rangle$ and $u \Downarrow_{\rho} G$ and $t' \Downarrow_{\rho'[x \leftarrow G]} F$. By Ind Hyp if $E \vdash t : \mathcal{G} \rightarrow \mathcal{F}$ and $\rho \models E$ and $t \Downarrow_{\rho} \langle \rho', x, t' \rangle$ then $\langle \rho', x, t' \rangle \models \mathcal{G} \rightarrow \mathcal{F}$. 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 8, $F \models \mathcal{F}$. \square

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 $\neg\mathbf{0}$.

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

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}{ll}
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} \rightarrow \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 [CG00, CG01b]. Model checking algorithms for various fragments without guarantee have been proposed [CDZG⁺01, CT01]. Lugiez

and Dal Zilio [LD02] 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 [Urq84]. Lincoln, Mitchell, Scedrov, and Shankar [LMSS92] prove undecidability for both propositional linear logic and propositional intuitionistic linear logic. See Cardelli and Gordon [CG00] for a detailed discussion of the differences between the ambient logic and relevant and linear logics.

Calcagno, Yang, and O’Hearn [CYO01] 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 [CT01] 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 [CM98] and Cardelli and Gordon [CG01b] 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 [Coh02] 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.

References

- [Bun97] P. Buneman. Semistructured data. In *16th ACM Symposium on Principles of Database Systems (PODS'97)*, 1997.
- [CC01] 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.
- [CC02] 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.
- [CDZG⁺01] 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.
- [CG00] 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.
- [CG01a] 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.
- [CG01b] 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.
- [CGG02] 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.
- [CM98] 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.
- [Coh02] E. Cohen. Validity and model checking for logics of finite multisets. Draft, Microsoft Research, 2002.
- [CT01] 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.
- [CYO01] 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.
- [HLS02] D. Hirschhoff, 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.
- [HP00] 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.
- [HP01] 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.
- [IO01] 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.
- [LD02] D. Lugiez and S. Dal Zilio. Multitrees automata, Presburger’s constraints and tree logics. Laboratoire d’Informatique Fondamentale, CNRS and Université de Provence, 2002.
- [LMSS92] P.D. Lincoln, J.C. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56:239–311, 1992.
- [ORY01] 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.
- [Rey02] J. C. Reynolds. Separation logic: a logic for shared mutable data structures. In *Logic in Computer Science (LICS’02)*, pages 55–74. IEEE, 2002.
- [Urq84] A. Urquhart. The undecidability of entailment and relevant implication. *Journal of Symbolic Logic*, 45:1059–1073, 1984.
- [XML] Extensible markup language. <http://www.w3.org/XML/>.