A Theory of Second-Order Trees

Neal Glew

January 2001

Abstract

This report describes a theory of second-order trees, that is, finite and infinite trees where nodes of the tree can bind variables that appear further down in the tree. Such trees can be used to provide a natural and intuitive interpretation for type systems with equirecursive types and binding constructs like universal and existential quantifiers. The report defines the set of *binding trees*, and a subset of these called *regular binding trees*. These are similar to the usual notion of regular trees, but generalised to take into account the binding structure. Then the report shows how to interpret a second-order type system with recursive quantifiers as binding trees, and gives a sound and complete axiomatisation of when two types map to the same tree. Finally the report gives a finite representation for trees called *tree automata*, and gives a construction for deciding when two automata map to the same tree. To tie everything together, the report defines a mapping from types to automata, thus giving a decision procedure for when two types map to the same tree.

1 Introduction

In the theory of type systems there are two approaches to recursive types, the *isorecursive* and *equirecursive* approach. In the *isorecursive approach*, the types $\mathsf{rec}\ \alpha.\tau$ and $\tau\{\alpha := \mathsf{rec}\ \alpha.\tau\}$[1] are considered different but isomorphic types. The expression language includes type coercions[2] $\mathsf{roll}(e)$ and $\mathsf{unroll}(e)$ for converting a value of one type to the other. The isorecursive approach is easier to construct decision procedures for, and is easier to prove sound; but it requires programs to contain type coercions. In the *equirecursive approach*, the types $\mathsf{rec}\ \alpha.\tau$ and $\tau\{\alpha := \mathsf{rec}\ \alpha.\tau\}$ are considered equal, and there are no expression-level constructs for dealing with recursive types. The presence of this equality makes it more difficult to develop decision procedures for equality and subtyping, and more difficult to prove the type system sound. However, there are no type coercions in programs, and more types are equivalent. For example, $\mathsf{rec}\ \alpha.\tau$ and $\mathsf{rec}\ \alpha.(\tau\{\alpha := \tau\})$ are equal in the equirecursive types approach but are not intercoercible in the isorecursive approach.

A more fundamental problem with the equirecursive approach is that previous work on formalising it has gaps (see below). This technical report fills these gaps by providing solid foundations for second-order type systems with equirecursive types.

[1] The notation $x\{y := z\}$ denotes the capture avoiding substitution of z for y in x.

[2] A type coercion changes the type of an expression but has no operational or runtime effect.

Amadio and Cardelli [AC93] were the first to investigate the equirecursive approach. They proposed the *tree interpretation* of recursive types, which is based on the idea that if the equality between $\mathsf{rec}\ \alpha.\tau$ and $\tau\{\alpha := \mathsf{rec}\ \alpha.\tau\}$ is applied repeatedly then recursive types expand into infinite trees that have no recursive quantifiers. Then, types are equal exactly when their fully-expanded trees are the same. Furthermore, subtyping can first be defined on trees and then lifted to types. Amadio and Cardelli defined a suitable tree model for a first-order type system, and a definition of subtyping between trees. Then they gave a set of type equality and subtyping rules, and proved them sound and complete with respect to the tree interpretation. Finally they gave an algorithm for deciding type equality and subtyping. Their algorithm is exponential time, which is much worse than the linear time algorithm in the isorecursive approach.

Kozen et al. [KPS95] reduced this exponential time complexity to polynomial time. First they defined term automata,[3] which, like types, are a finite representation for trees. Briefly, a term automata is finite state machine. Each state represents a collection of nodes in a tree, the initial state is the root of the tree, and the transition function gives the children for each node and the labels on the edges to these children. Kozen et al. gave an intersection-like construction on term automata that can be used to decide equality and subtyping in quadratic time. In both Amadio and Cardelli and Kozen et al.'s work, only first-order systems were considered, and their results do not generalise in a straightforward manner to second-order types.

Colazzo and Ghelli [CG99] investigated a second-order type system with equirecursive quantifiers. They gave a coinductively defined set of type rules (type rules are normally defined inductively), and described and proved correct an algorithm for deciding subtyping. They did not show the relation between their rules and Amadio and Cardelli's rules. Nor did they analyse the complexity of their algorithm, although they conjectured it was exponential. Their algorithm is essentially a search algorithm with the curious feature that it cuts off search when it sees the same subtyping judgement for the third, not second, time. They were able to show that this criterion is necessary and sufficient, but never gave an intuitive explanation.

This technical report extends the tree interpretation and idea of tree automata from first-order to second-order trees. The first contribution is a notion of second-order finite and infinite trees suitable as a semantic model for types. The second contribution is a proof that the usual equality rules for equirecursive types are sound and complete for this model. The third contribution is notion of tree automata suitable for second-order trees. The fourth contribution is a polynomial time decision procedure for type equality. This technical report deals only with equality and subtyping is left to future work. Since equality is not something specific to types, I will call them terms in the sequel. The rest of the technical report presents each contribution in turn. This technical report is an extended version of a conference paper [Gle02], and it contains full details including proofs.

2 Preliminaries

The theory is meant to be general and to abstract over everything else in the term system. Therefore, I will assume the term language consists of variables, recursive quantifiers, and terms

[3] This technical report will call Kozen et al.'s term automata, tree automata.

build from node labels $nl \in NL$. Each node label will take a number of arguments, which will be identified using labels $\ell \in L$, and for each argument may bind a certain number of variables. The function $spec \in NL \to L \overset{\text{fin}}{\rightharpoonup} \mathbb{N}$ defines which arguments a node label takes and how many variables it binds for each. For example, the system $\mathcal{F}_{\leq\mu}$ of Colazzo and Ghelli has $NL = \{\top, \to, \forall\}$, $L = \{\mathsf{arg}, \mathsf{bnd}, \mathsf{bdy}, \mathsf{res}\}$, and $spec = \{(\to, \mathsf{arg}, 0), (\to, \mathsf{res}, 0), (\forall, \mathsf{bnd}, 0), (\forall, \mathsf{bdy}, 1)\}$. The rest of the report will not refer to $spec$, instead it will use functions $\textsc{labels}(nl) = \mathrm{dom}(spec(nl))$ and $\textsc{bind}(nl, \ell) = spec(nl)(\ell)$.

Notations and Conventions $A \rightharpoonup B$ is the set of all partial functions from A to B; $A \overset{\text{fin}}{\rightharpoonup} B$ is the set of partial functions from A to B with finite domain; $A \overset{c}{\to} B$ is the set of functions from A to B that are c-Lipschitz. If e_1 and e_2 are possibly undefined expressions then $e_1 = e_2$ means that either both are defined and equal or both are undefined. A^* is the set of finite sequences of elements from A; ϵ is the empty sequence; prefix, concatenation, and append are written using juxtaposition; if $x, y \in A^*$ then $x \leq y$ means that x is a prefix of y. If P is a set of sequences then P^* is the usual Kleene operator, that is, $\{p_1 \cdots p_n \mid n \geq 0 \land p_i \in P\}$; if P_1 and P_2 are sets of sequences then $P_1 P_2 = \{p_1 p_2 \mid p_1 \in P_1 \land p_2 \in P_2\}$. $|\cdot|$ is used both as the size of a set and the length of a sequence. $A + B$ is the disjoint union of A and B; $\mathrm{in}_i(x)$ is the injection of x into the ith arm of a disjoint union. If f is a function then $f\{x \mapsto y\}$ is the same function except that it maps x to y. If R is an equivalence relation then $[R]$ is the set of its equivalence classes and $[x]_R$ is the equivalence class of x under R. If R is a relation then R^* is its transitive reflexive closure and R^+ is its transitive closure.

3 Binding Trees

Binding trees are just finite or infinite trees whose nodes are labeled by node labels or by variables in a De Bruijn representation [Bru72] and whose edges are labeled by L. A node labeled nl has edges labeled by $\textsc{labels}(nl)$ and variables are leafs. This can be formalised as follows.

Definition 3.1 *The set of binding trees is defined as:*

$$Tree = \{t : L^* \rightharpoonup NL + \mathbb{N} \mid \epsilon \in \mathrm{dom}(t) \land (p\ell \in \mathrm{dom}(t) \Leftrightarrow \exists nl : t(p) = \mathrm{in}_1(nl) \land \ell \in \textsc{labels}(nl))\}$$

Distance between two trees is defined as $d(t_1, t_2) = 2^{-\min\{n \mid t_1(p) \neq t_2(p) \land n = |p|\}}$ *where* $2^{-\min \emptyset} = 0$.

Recursive quantifiers will be interpreted as fixed points of maps on $Tree$, so it is important that such fixed points exist. In fact, $(Tree, d)$ is a complete ultrametric space, which means that contractive maps on it have unique fixed points (c.f. [Kap77]).

Theorem 3.1 $(Tree, d)$ *is a complete ultrametric space.*

Proof: First, $d(t_1, t_2) = 0 \Leftrightarrow t_1 = t_2$:

$$\begin{array}{ll} & d(t_1,t_2) = 0 \\ \Leftrightarrow & \{n \mid n = |p| \wedge t_1(p) \neq t_2(p)\} = \emptyset \\ \Leftrightarrow & \forall p \in L^* : t_1(p) = t_2(p) \\ \Leftrightarrow & t_1 = t_2 \end{array}$$

Second, $d(t_1, t_2) = d(t_2, t_1)$: This holds by inspection of the definition.

Third, $d(t_1, t_3) \leq \max(d(t_1, t_2), d(t_2, t_3))$: Clearly $t_1(p) = t_2(p) \wedge t_2(p) = t_3(p) \Rightarrow t_1(p) = t_3(p)$ for all $p \in L^*$. Then:

$$\begin{array}{ll} & \forall p \in L^* : p \notin \{p \mid t_1(p) \neq t_2(p)\} \wedge p \notin \{p \mid t_2(p) \neq t_3(p)\} \Rightarrow p \notin \{p \mid t_1(p) \neq t_3(p)\} \\ \Rightarrow & \{p \mid t_1(p) \neq t_3(p)\} \subseteq \{p \mid t_1(p) \neq t_2(p)\} \cup \{p \mid t_2(p) \neq t_3(p)\} \\ \Rightarrow & \{n \mid n = |p| \wedge t_1(p) \neq t_3(p)\} \subseteq \\ & \quad \{n \mid n = |p| \wedge t_1(p) \neq t_2(p)\} \cup \{n \mid n = |p| \wedge t_2(p) \neq t_3(p)\} \\ \Rightarrow & \min\{n \mid n = |p| \wedge t_1(p) \neq t_3(p)\} \geq \\ & \quad \min(\{n \mid n = |p| \wedge t_1(p) \neq t_2(p)\} \cup \{n \mid n = |p| \wedge t_2(p) \neq t_3(p)\}) \\ \Rightarrow & -\min\{n \mid n = |p| \wedge t_1(p) \neq t_3(p)\} \leq \\ & \quad \max(-\min\{n \mid n = |p| \wedge t_1(p) \neq t_2(p)\}, -\min\{n \mid n = |p| \wedge t_2(p) \neq t_3(p)\}) \\ \Rightarrow & 2^{-\min\{n|n=|p| \wedge t_1(p) \neq t_3(p)\}} \leq \max(2^{-\min\{n|n=|p| \wedge t_1(p) \neq t_2(p)\}}, 2^{-\min\{n|n=|p| \wedge t_2(p) \neq t_3(p)\}}) \\ \Rightarrow & d(t_1, t_3) \leq \max(d(t_1, t_2), d(t_2, t_3)) \end{array}$$

These three facts establish that (Tree, d) is an ultrametric space. Consider if it is complete. Let $\langle t_i \rangle_{i \in \mathbb{N}}$ be a Cauchy sequence of trees, in particular, for $\delta \in (0, 1]$, let N_δ be such that $d(t_n, t_m) < \delta$ for all $n, m \geq N_\delta$; without loss of generality assume that $N_{\delta_1} \geq N_{\delta_2}$ when $\delta_1 \leq \delta_2$. Let M_n ne $N_{2^{-n}}$ and t be $\lambda p.t_{M_{|p|}}(p)$. First, $t \in \mathit{Tree}$: $\epsilon \in \mathrm{dom}(t)$ as $t(\epsilon) = t_{M_0}(\epsilon)$ and $t_{M_0} \in \mathit{Tree}$. Now consider $p\ell$, $p\ell \in \mathrm{dom}(t)$ exactly when $p\ell \in \mathrm{dom}(t_{M_{|p\ell|}})$. By the definition of *Tree*, this occurs exactly when $t_{M_{|p\ell|}}(p) = \mathrm{in}_1(nl) \wedge \ell \in \textsc{labels}(nl)$. By the definition of M and N, $d(t_{M_{|p|}}, t_{M_{|p\ell|}}) < 2^{-|p|}$. By the definition of distance, $\min\{|p'| \mid t_{M_{|p|}}(p') \neq t_{M_{|p\ell|}}\} > |p|$. Therefore $t_{M_{|p|}}(p) = t_{M_{|p\ell|}}(p)$ so $t(p) = t_{M_{|p\ell|}}(p)$. Thus $p\ell \in \mathrm{dom}(t) \Leftrightarrow t(p) = \mathrm{in}_1(nl) \wedge \ell \in \textsc{labels}(nl)$ as required. Second, $\lim \langle t_i \rangle_{i \in \mathbb{N}} = t$: Let $\epsilon \in (0, 1]$ and let $n \geq M_i$ where $2^{-i} \leq \epsilon$. Consider a path p such that $|p| \leq i$, so $n \geq M_i \geq M_{|p|}$. By definition, $d(t_{M_{|p|}}, t_n) < 2^{-|p|}$, so $t(p) = t_{M_{|p|}}(p) = t_n(p)$. Thus $d(t, t_n) < 2^{-i} \leq \epsilon$, as required. □

The following two tree constructors will be needed latter. The first builds a tree whose root is the free variable n, and the second builds a tree with root nl and the given trees as the children of the root.

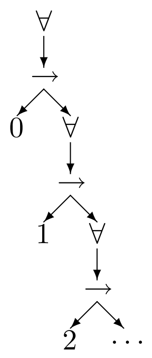
Definition 3.2 *Tree* $\mathsf{var}(n) = \{(\epsilon, \mathrm{in}_2(n))\}$ *and tree* $\mathsf{nl}(nl, \ell = t_{\ell \in \mathrm{LABELS}(nl)}) = \{(\epsilon, \mathrm{in}_1(nl))\} \cup \{(\ell p, t_\ell(p)) \mid \ell \in \textsc{labels}(nl) \wedge p \in \mathrm{dom}(t_\ell)\}$

These do in fact produce trees.

Lemma 3.2 *If* $n \in \mathbb{N}$ *then* $\mathsf{var}(n) \in \mathit{Tree}$. *If* $nl \in NL$ *and* $t_\ell \in \mathit{Tree}$ *for* $\ell \in \textsc{labels}(nl)$ *then* $\mathsf{nl}(nl, \ell = t_\ell) \in \mathit{Tree}$.

Proof: The result follows straightfowardly from the definitions. □

Not all trees are expressible using terms or automata, so it is important to define a subset of trees that are. In the theory of first-order trees, these are called regular trees, and are defined as those with a finite number of subtrees. This definition is inadequate for binding trees, as the term $\forall\alpha.\mathsf{rec}\ \beta.(\alpha \to \forall\gamma.\beta)$ maps to the following tree



which has a subtree of the form $\mathsf{var}(n)$ for each $n \in \mathbb{N}$. The problem is that the De Bruijn indices represent conceptually the same variable as possibility different tree nodes. The definition of regular trees needs to take into account the binding structure and De Bruijn representation. Before giving the definition, I need to develop a number of other basic concepts.

3.1 Binding

The $\textsc{bind}(\cdot,\cdot)$ function can be extended to paths within trees in a natural way.

Definition 3.3 *The binding of a path p in a tree t, $\textsc{bind}(t,p)$, is defined by:*

$$\begin{array}{rcl} \textsc{bind}(t,\epsilon) & = & 0 \\ \textsc{bind}(t,p\ell) & = & \textsc{bind}(t,p) + \textsc{bind}(nl,\ell) \ \textit{where}\ t(p) = \mathrm{in}_1(nl) \end{array}$$

The binding from path p_1 to path p_2 in a tree t, $\textsc{bind}(t,p_1 \to p_2) = \textsc{bind}(t,p_2) - \textsc{bind}(t,p_1)$.

Lemma 3.3 *If $t \in \mathit{Tree}$ then $\textsc{bind}(t,p)$ is defined exactly when $p \in \mathrm{dom}(t)$. If $t \in \mathit{Tree}$, $p_1, p_2 \in \mathrm{dom}(t)$, and $p_1 < p_2$ then $\textsc{bind}(t,p_1 \to p_2)$ is defined.*

Proof: The result follows straightforwardly from the definitions. □

3.2 Identified Variable

The key to regular trees is that variable nodes that conceptually refer to the same variable (e.g., the same free variable or are bound by the same node) should be considered equal. The next definition makes precise what a variable node in a tree refers to.

Definition 3.4 *The variable identified by path p of tree t where $t(p) = \mathrm{in}_2(n)$ is:*

$$\textsc{varof}(t,p) = \begin{cases} \mathsf{free}(n - \textsc{bind}(t,p)) & \textsc{bind}(t,p) \le n \\ \mathsf{bound}(p_1, \ell, i) & p_1\ell \le p \wedge \textsc{bind}(t, p_1\ell \to p) \le n < \textsc{bind}(t, p_1 \to p) \wedge \\ & i = n - \textsc{bind}(t, p_1\ell \to p) \end{cases}$$

otherwise $\textsc{varof}(t,p)$ *is undefined. (Here* free *and* bound *should be thought of as tags in a disjoint union that are used to distinguish the two possible results of the function.)*

This definition satisfies a number of basic properties that will be needed in later proofs.

Lemma 3.4

- *If $t \in$ Tree and $p \in L^*$ then $\textsc{varof}(t,p)$ is defined exactly when $t(p) = \mathrm{in}_2(n)$, if $\textsc{varof}(t,p)$ is defined then it is uniquely defined, and if $\textsc{varof}(t,p) = \mathsf{bound}(p', \ell, i)$ then $p' < p$, $t(p') = \mathrm{in}_1(nl)$, and $0 \le i < \textsc{bind}(nl, \ell)$.*
- *If $t \in$ Tree and $p \in \mathrm{dom}(t)$ then exactly one of the following holds: $t(p) = \mathrm{in}_1(nl)$, $\textsc{varof}(t,p) = \mathsf{free}(n)$, or $\textsc{varof}(t,p) = \mathsf{bound}(p', \ell, n)$. Furthermore the nl, n, p', and ℓ are uniquely determined.*
- *If $t \in$ Tree then: $\textsc{varof}(t,p) = \mathsf{free}(n)$ if and only if $t(p) = \mathrm{in}_2(n + \textsc{bind}(t,p))$.*
- *If $t \in$ Tree then: $\textsc{varof}(t,p) = \mathsf{bound}(p', \ell, n)$ if and only if $t(p) = \mathrm{in}_2(n + \textsc{bind}(t, p'\ell \to p))$, $t(p') = \mathrm{in}_1(nl)$, and $0 \le n < \textsc{bind}(nl, \ell)$.*

Proof: The results follow straightforwardly from the definitions and, in some cases, by induction on the paths in question. □

3.3 Free Variables

Given the previous definition, the free variables of a tree are easily defined.

Definition 3.5 *The free variables of a tree t are $\textsc{ftv}(t) = \{n \mid \textsc{varof}(t,p) = \mathsf{free}(n)\}$.*

3.4 Shifting

The *shifting* operation is important to De Bruijn representations. It returns an identical tree except that the tree's free variables are incremented by a constant. Later a more specialised shifting operation is needed, it increments only free variables above a certain index.

Definition 3.6 *The shift of a tree t above m by n is:*

$$\textsc{shift}(t,m,n) = \lambda p. \begin{cases} \mathrm{in}_2(n+i) & t(p) = \mathrm{in}_2(i) \wedge i \ge \textsc{bind}(t,p) + m \\ t(p) & \textit{otherwise} \end{cases}$$

The shift of a tree t by n is $\textsc{shift}(t,n) = \textsc{shift}(t,0,n)$.

The shifting operation satisfies a number of basic properties that will be needed in later proofs.

Lemma 3.5

- *If* $t \in \mathit{Tree}$ *and* $m, n \in \mathbb{N}$ *then* $\textsc{shift}(t, m, n) \in \mathit{Tree}$ *and* $\textsc{shift}(t, n) \in \mathit{Tree}$*.*
- $\mathrm{dom}(t) = \mathrm{dom}(\textsc{shift}(t, m, n)) = \mathrm{dom}(\textsc{shift}(t, n))$
- $\textsc{bind}(\textsc{shift}(t, n), p) = \textsc{bind}(t, p)$
- *If* $p \in \mathrm{dom}(t)$ *then either* $t(p) = \mathrm{in}_1(nl)$ *and* $\textsc{shift}(t, n)(p) = \mathrm{in}_1(nl)$*,* $\textsc{varof}(t, p) = \mathsf{free}(i)$ *and* $\textsc{varof}(\textsc{shift}(t, n), p) = \mathsf{free}(n + i)$*, or* $\textsc{varof}(t, p) = \mathsf{bound}(p', \ell, i)$ *and* $\textsc{varof}(\textsc{shift}(t, n), p) = \mathsf{bound}(p', \ell, i)$*.*
- $\textsc{ftv}(\textsc{shift}(t, n)) = \{n + i \mid i \in \textsc{ftv}(t)\}$
- $\textsc{shift}(\textsc{shift}(t, m_1, n_1), m_2, n_2) = \textsc{shift}(t, m_1, n_1 + n_2)$ *if* $m_1 \leq m_2 \leq m_1 + n_1$
- $\textsc{shift}(\textsc{shift}(t, n_1), n_2) = \textsc{shift}(t, n_1 + n_2)$
- $d(\textsc{shift}(t_1, n), \textsc{shift}(t_2, n)) = d(t_1, t_2)$
- $\textsc{shift}(\mathsf{var}(m), n) = \mathsf{var}(m + n)$
- $\textsc{shift}(\mathsf{nl}(nl, \ell = t_\ell), m, n) = \mathsf{nl}(nl, \ell = \textsc{shift}(t_\ell, m + \textsc{bind}(nl, \ell), n))$

Proof: The first two items follow straightforwardly from the definitions. The third item follows from the definitions by induction on p. The fourth item follows from the definitions and the second and third item. The fifth item is immediate from the fourth. For the sixth item, let $t_1 = \textsc{shift}(t, m_1, n_1)$ then:

$$\textsc{shift}(t_1, m_2, n_2)(p) = \begin{cases} \mathrm{in}_2(i + n_1 + n_2) & t(p) = \mathrm{in}_2(i) \wedge i \geq \textsc{bind}(t, p) + m_1 \wedge \\ & i + n_1 \geq \textsc{bind}(t_1, p) + m_2 \\ \mathrm{in}_2(i + n_2) & t(p) = \mathrm{in}_2(i) \wedge i < \textsc{bind}(t, p) + m_1 \wedge \\ & i \geq \textsc{bind}(t_1, p) + m_2 \\ \mathrm{in}_2(i + n_1) & t(p) = \mathrm{in}_2(i) \wedge i \geq \textsc{bind}(t, p) + m_1 \wedge \\ & i + n_1 < \textsc{bind}(t_1, p) + m_2 \\ t(p) & \text{otherwise} \end{cases}$$

Since $m_1 \leq m_2$ the condition in the second alternative is impossible. Since $m_1 + n_1 \geq m_2$, by the third item, the condition in the first alternative reduces to $t(p) = \mathrm{in}_2(i) \wedge i \geq \textsc{bind}(t, p) + m_1$, and the condition on the third alternative is impossible. Thus

$$\textsc{shift}(t_1, m_2, n_2)(p) = \begin{cases} \mathrm{in}_2(i + n_1 + n_2) & t(p) = \mathrm{in}_2(i) \wedge i \geq \textsc{bind}(t, p) + m_1 \\ t(p) & \text{otherwise} \end{cases}$$

which is equal to the definition of $\textsc{shift}(t, m_1, n_1 + n_2)$. The seventh item follows from the fourth item and the definitions. The eighth item follows from the second and fourth item. The last two items follow straightforwardly from the definitions. $\square$

3.5 Subtrees

The subtree of a tree along some path is easily defined.

Definition 3.7 *The subtree of t at p is* $\textsc{subtree}(t,p) = \lambda p'.t(pp')$. *The subtrees of t are* $\textsc{subtrees}(t) = \{\textsc{subtree}(t,p) \mid p \in \mathrm{dom}(t)\}$.

Some properties of subtrees will be needed in later proofs.

Lemma 3.6

- *If $t \in \mathit{Tree}$ and $p \in \mathrm{dom}(t)$ then $\textsc{subtree}(t,p) \in \mathit{Tree}$.*
- *If $t \in \mathit{Tree}$ then $\textsc{subtrees}(t) \subseteq \mathit{Tree}$.*
- *If $t(p) = \mathrm{in}_1(nl)$ then $\textsc{subtree}(t,p) = \mathsf{nl}(nl, \ell = \textsc{subtree}(t, p\ell))$.*
- *If $t \in \mathit{Tree}$ then: $\textsc{varof}(t,p) = \mathsf{free}(n)$ if and only if $\textsc{subtree}(t,p) = \mathsf{var}(n + \textsc{bind}(t,p))$.*
- *If $t \in \mathit{Tree}$ then: $\textsc{varof}(t,p) = \mathsf{bound}(p', \ell, n)$ if and only if $\textsc{subtree}(t,p) = \mathsf{var}(n + \textsc{bind}(t, p'\ell \to p))$, $t(p') = \mathrm{in}_1(nl)$, and $0 \leq n < \textsc{bind}(nl, \ell)$.*
- *If $pp' \in \mathrm{dom}(t)$ then $\textsc{bind}(\textsc{subtree}(t,p), p') = \textsc{bind}(t, p \to p')$.*
- *If $pp_2 \in \mathrm{dom}(t)$ and $p_1 \leq p_2$ then $\textsc{bind}(\textsc{subtree}(t,p), p_1 \to p_2) = \textsc{bind}(t, pp_1 \to pp_2)$.*
- $\textsc{subtree}(\textsc{subtree}(t, p_1), p_2) = \textsc{subtree}(t, p_1 p_2)$
- $\textsc{subtree}(\mathsf{nl}(nl, \ell = t_\ell), \ell) = t_\ell$

Proof: The first, third, and sixth through nineth items follow straightforwardly from the definitions. The second item follows from the first. The fourth and fifth items follow from the definitions and Lemma 3.4. □

3.6 Substitution

There are a number of ways to substitute trees into other trees, this section defines each of them.

Definition 3.8 *The capture avoiding substitution of t_2 for path p in t_1 is:*

$$t_1\{p := t_2\} = \lambda p'. \begin{cases} \textsc{shift}(t_2, \textsc{bind}(t_1, p))(p'') & p' = pp'' \\ t_1(p') & p \not\leq p' \end{cases}$$

A set of paths P is incomparable if $p_1, p_2 \in P$ and $p_1 \leq p_2$ implies $p_1 = p_2$. The simultaneous substitution of t_i for p_i in t, where $\{p_i \mid i \in I\}$ is incomparable, is:

$$t\{p_i := t_i\}_{i \in I} = \lambda p. \begin{cases} \textsc{shift}(t_i, \textsc{bind}(t, p_i))(p') & p = p_i p' \\ t(p) & \forall i \in I : p_i \not\leq p \end{cases}$$

The substitution of t_2 for free variable n in t_1 is $t_1\{n := t_2\} = t_1\{p := t_2\}_{\text{VAROF}(t,p)=\mathsf{free}(n)}$*.*

A substitution (metavariable ρ) is a function in $\mathbb{N} \to$ Tree. The substitution of ρ in t is $t\{\rho\} = t\{p := \rho(n)\}_{\text{VAROF}(t,p)=\mathsf{free}(n)}$*. The shift of a substitution is:*

$$\text{SHIFT}(\rho, n) = \lambda i. \begin{cases} \mathsf{var}(i) & i < n \\ \text{SHIFT}(\rho(i-n), n) & i \geq n \end{cases}$$

All these notions of substitution are well defined.

Lemma 3.7

- *If $t_1, t_2 \in$ Tree and $p \in \mathrm{dom}(t_1)$ then $t_1\{p := t_2\} \in$ Tree.*
- *If $t \in$ Tree, $t_i \in$ Tree for $i \in I$, $p_i \in \mathrm{dom}(t)$ for $i \in I$, and $\{p_i \mid i \in I\}$ is incomparable then $t\{p_i := t_i\}_{i \in I} \in$ Tree.*
- *If $t_1, t_2 \in$ Tree then $t_1\{n := t_2\} \in$ Tree.*
- *If ρ is a substitution and $t \in$ Tree then $t\{\rho\} \in$ Tree.*
- *If ρ is a substitution then $\text{SHIFT}(\rho, n)$ is a substitution.*

Proof: The results follow straightforwardly from the definitions. □

Substitution satisfies several basic properties that are needed in later proofs.

Lemma 3.8

- $t_1\{P_1 := t_2\{P_2 := t_3\}\} = t_1\{P_1 := t_2\}\{P_1P_2 := t_3\}$
- *If $P_1 \cup P_2$ is incomparable then $t_1\{P_1 := t_2\}\{P_2 := t_2\} = t_1\{P_1 \cup P_2 := t_2\}$*
- $\mathsf{nl}(nl, \ell = t_\ell\{P_\ell := \text{SHIFT}(t, \text{BIND}(nl, \ell))\}) = \mathsf{nl}(nl, \ell = t_\ell)\{\{\ell p \mid p \in P_\ell\} := t\}$
- $\text{SHIFT}(t_1\{P := t_2\}, m, n) = \text{SHIFT}(t_1, m, n)\{P := \text{SHIFT}(t_2, m, n)\}$

Proof: The results follow straightforwardly from the definitions. □

3.7 Regular Trees

Using the machinery so far, when two subtrees of a tree might be equal is defined coinductively as follows.

Definition 3.9 *A relation R is an equivalence of t's subtrees if and only if R is an equivalence relation on $\mathrm{dom}(t)$ and $p_1 \mathrel{R} p_2$ implies $\text{EQSTPROP}(p_1, R, p_2)$ where the latter is either*

- $t(p_1) = \mathrm{in}_1(nl)$, $t(p_2) = \mathrm{in}_1(nl)$, *and* $p_1\ell \; R \; p_2\ell$ *for all* $\ell \in \textsc{labels}(nl)$,
- $\textsc{varof}(t, p_1) = \mathsf{free}(n)$ *and* $\textsc{varof}(t, p_2) = \mathsf{free}(n)$, *or*
- $\textsc{varof}(t, p_1) = \mathsf{bound}(p'_1, \ell, n)$, $\textsc{varof}(t, p_2) = \mathsf{bound}(p'_2, \ell, n)$, *and* $p'_1 \; R \; p'_2$.

However, more is needed. The term $\mathsf{rec}\ \alpha.\sigma$ will expand into a repetitive tree that has many copies of the tree corresponding to σ. Bound variables that appear in a copy of this tree always bind to a binder within that same copy, never to a binder in a different copy of the tree. The definition of equivalence of subtrees does not rule out binding across copies of repeating trees.

Definition 3.10 *An equivalence R of t's subtrees is nonoverlapping exactly when if $p_1 < p_2 < p_3$ and $\textsc{varof}(t, p_3) = \mathsf{bound}(p_1, \ell, n)$ then $(p_1, p_2) \notin R$.*

A tree t is regular if there exists a nonoverlapping equivalence of t's subtrees with a finite number of equivalence classes. RegTree is the set of regular trees.

While the definition of regular binding trees is not nearly as elegant as that for regular trees without binding, it nevertheless does define exactly the set of trees that are expressible by terms or automata, as will be proven later.

It turns out that trees have a greatest nonoverlapping equivalence of their subtrees; later sections will use this to define a canonical term for each tree and to prove completeness of the equality rules.

Definition 3.11 *For $t \in$ Tree, $\textsc{eqst}(t)$ is the union of the nonoverlapping equivalences of t's subtrees.*

Some basic properties are needed for the proof of the next lemma and the proof of completeness.

Lemma 3.9

- *If R is an equivalence of t's subtrees, $p_1 \; R \; p_2$, and $p_1 p \in \mathrm{dom}(t)$ then $p_2 p \in \mathrm{dom}(t)$ and $p_1 p \; R \; p_2 p$.*
- *If R is an equivalence of t's subtrees, $p_1 < p_2$, $p_1 \; R \; p_2$, and $\neg\exists p'_1, p'_2 : p'_1 < p'_2 < p_2 \wedge p'_1 \; R \; p'_2$ then $\textsc{subtree}(t, p_2) = \textsc{shift}(\textsc{subtree}(t, p_1), \textsc{bind}(t, p_1 \to p_2))$.*
- *If $p_2 \in \mathrm{dom}(t)$, $p_1 < p_2$, and $\textsc{subtree}(t, p_2) = \textsc{shift}(\textsc{subtree}(t, p_1), \textsc{bind}(t, p_1 \to p_2))$ then $p_1 \; \textsc{eqst}(t) \; p_2$.*

Proof: The first item is a straightforward induction on p and the property of being an equivalence of subtrees. For the second item, let $t_1 = \textsc{shift}(\textsc{subtree}(t, p_1), \textsc{bind}(t, p_1 \to p_2))$. By the first item and Lemma 3.5, $\mathrm{dom}(\textsc{subtree}(t, p_2)) = \mathrm{dom}(\textsc{subtree}(t, p_1)) = \mathrm{dom}(t_1)$ and $p_1 p \; R \; p_2 p$ for all p in this domain. It suffices to show that $\textsc{subtree}(t, p_2)(p) = t_1(p)$ for all such p. By an easy induction, $\textsc{bind}(t, p_1 \to p_1 p) = \textsc{bind}(t, p_2 \to p_2 p)$ $(*)$. Case 1,

$t(p_2p) = \mathrm{in}_1(nl)$: By definition of equivalence of subtrees, $t(p_1p) = \mathrm{in}_1(nl)$. By definition, $\textsc{subtree}(t, p_1)(p) = \mathrm{in}_1(nl)$ and again by definition, $t_1(p) = \mathrm{in}_1(nl)$, as required. Case 2, $\textsc{varof}(t, p_2p) = \mathsf{free}(n)$: By definition of equivalence of subtrees, $\textsc{varof}(t, p_1p) = \mathsf{free}(n)$. By Lemma 3.4, $\textsc{subtree}(t, p_2)(p) = \mathrm{in}_2(n + \textsc{bind}(t, p_2p))$ and $\textsc{subtree}(t, p_1)(p) = \mathrm{in}_2(n + \textsc{bind}(t, p_1p))$, so By definition, $n + \textsc{bind}(t, p_1p) \geq \textsc{bind}(t, p_1 \to p_1p)$. By Lemma 3.6, the latter is $\textsc{bind}(\textsc{subtree}(t, p_1), p)$. By the definition of shift, $t_1(p) = \mathrm{in}_2(n+\textsc{bind}(t, p_1p)+\textsc{bind}(t, p_1 \to p_2))$. By $(*)$ and definition, $t_1(p) = \mathrm{in}_2(n + \textsc{bind}(t, p_2p))$, as required. Case 3, $\textsc{varof}(t, p_2p) = \mathsf{bound}(p'_1, \ell, n)$: By the definition of equivalence of subtrees, $\textsc{varof}(t, p_1p) = \mathsf{bound}(p'_2, \ell, n)$ and $p'_1\ R\ p'_2$. By Lemma 3.4, $t(p_ip) = \mathrm{in}_2(n + \textsc{bind}(t, p'_i\ell \to p_ip))$ $(**)$. Consider the cases for p'_1 and p'_2. If $p'_1 = p_1p'$ then $p'_2\ R\ p_2p'$. Since R is nonoverlapping this means that $p'_2 \geq p_2p'$. Similarly, if $p'_2 = p_2p'$ then $p'_1 \geq p_1$. If $p'_i = p_ip''_i$ then $p'_i\ R\ p_ip''_{1-i}$. Since R is nonoverlapping it must be that $p''_1 = p''_2$. Thus one possibility is that $p'_i = p_ip'$. If $p'_1 < p_1$ and $p'_2 \leq p_1p, p_2$ then since R is nonoverlapping $p'_1 = p'_2$. Thus another possibility is that $p'_1 = p'_2 < p_1$. If $p'_1 < p_1$, $p'_2 < p_2$, and $p'_2 \not\leq p_1p$ then $p'_1 < p'_2 < p_2$. But since $p'_1\ R\ p'_2$ this contradicts the assumption that $\neg\exists p'_1, p'_2 : p'_1 < p'_2 < p_2 \wedge p'_1\ R\ p'_2$; so the above two possibilities are the only possibilities. Subcase 1, $p'_i = p_ip'$: By Lemma 3.6, $\textsc{bind}(\textsc{subtree}(t, p_1), p) = \textsc{bind}(t, p_1 \to p_1p)$. Since $p_1 < p'_1\ell$ and $n < \textsc{bind}(nl, \ell)$ where $t(p'_1) = \mathrm{in}_1(nl)$, $n + \textsc{bind}(t, p'_1\ell \to p_1p) < \textsc{bind}(\textsc{subtree}(t, p_1), p)$. Thus the shift in t_1 has no effect, and by $(**)$, $t_1(p) = \mathrm{in}_2(n + \textsc{bind}(t, p_1p'\ell \to p_1p))$. By $(*)$ and definition, this equals $\mathrm{in}_2(n + \textsc{bind}(t, p_2p'\ell \to p_2p))$. By $(**)$, $\textsc{subtree}(t, p_2)(p) = \mathrm{in}_2(n + \textsc{bind}(t, p_2p'\ell \to p_2p))$, as required. Subcase 2, $p'_1 = p'_2 < p_1$: By $(**)$, $\textsc{subtree}(t, p_2)(p) = \mathrm{in}_2(n + \textsc{bind}(t, p'_1\ell \to p_2p))$ and $\textsc{subtree}(t, p_1)(p) = \mathrm{in}_2(n + \textsc{bind}(t, p'_1\ell \to p_1p))$. In this case the shift in t_1 has effect, so $t_1(p) = \mathrm{in}_2(n + \textsc{bind}(t, p'_1\ell \to p_1p) + \textsc{bind}(t, p_1 \to p_2))$. By $(*)$ and definition, the latter equals $\mathrm{in}_2(n + \textsc{bind}(t, p'_1\ell \to p_2p))$, as required.

For the third item, by an easy induction, $\textsc{bind}(t, p_1 \to p_1p) = \textsc{bind}(t, p_2 \to p_2p)$ $(*)$. Let $t_1 = \textsc{shift}(\textsc{subtree}(t, p_1), \textsc{bind}(t, p_1 \to p_2))$ and $R = \{(p, p) \mid p \in \mathrm{dom}(t)\} \cup \{(p_1p, p_2p), (p_2p, p_1p) \mid p_1p \in \mathrm{dom}(t)\}$. First note that by Lemma 3.5, $\mathrm{dom}(\textsc{subtree}(t, p_1)) = \mathrm{dom}(\textsc{subtree}(t, p_2))$ so $p_1p \in \mathrm{dom}(t)$ implies that $p_2p \in \mathrm{dom}(t)$. Thus R is an equivalence relation on $\mathrm{dom}(t)$. Let p be such that $p_1p \in \mathrm{dom}(t)$ to show that $\textsc{eqstprop}(p_1p, R, p_2p)$ ($\textsc{eqstprop}(p, R, p)$ is trivial and $\textsc{eqstprop}(p_2p, R, p_1p)$ is symmetric). Case 1, $t(p_2p) = \mathrm{in}_1(nl)$: By definition, $\textsc{subtree}(t, p_2)(p) = \mathrm{in}_1(nl)$, so $t_1(p) = \mathrm{in}_1(nl)$, so by definition, $\textsc{subtree}(t, p_1) = \mathrm{in}_1(nl)$. Thus by definition, $t(p_1p) = \mathrm{in}_1(nl)$. If $\ell \in \textsc{labels}(nl)$ then by definition of trees, $p_1p\ell \in \mathrm{dom}(t)$, so $p_1p\ell\ R\ p_2p\ell$, as required. Case 2, $\textsc{varof}(t, p_2p) = \mathsf{free}(n)$: By Lemma 3.4 and definition, $\textsc{subtree}(t, p_2)(p) = \mathrm{in}_2(n + \textsc{bind}(t, p_2p))$. Thus $t_1(p) = \mathrm{in}_2(n + \textsc{bind}(t, p_2p))$. Since $n + \textsc{bind}(t, p_2p) \geq \textsc{bind}(\textsc{subtree}(t, p_1), p)$, by definition of shift, $\textsc{subtree}(t, p_1)(p) = \mathrm{in}_2(n + \textsc{bind}(t, p_2p) - \textsc{bind}(t, p_1 \to p_2))$. By definition and $(*)$, $t(p_1p) = \mathrm{in}_2(n + \textsc{bind}(t, p_1) + \textsc{bind}(t, p_2 \to p_2p)) = \mathrm{in}_2(n + \textsc{bind}(t, p_1) + \textsc{bind}(t, p_1 \to p_1p)) = \mathrm{in}_2(n + \textsc{bind}(t, p_1p))$. By Lemma 3.4, $\textsc{varof}(t, p_1p) = \mathsf{free}(n)$, as required. Case 3, $\textsc{varof}(t, p_2p) = \mathsf{bound}(p', \ell, n)$: By Lemma 3.4 and definition, $t(p') = \mathrm{in}_1(nl)$, $0 \leq n < \textsc{bind}(nl, \ell)$, and $\textsc{subtree}(t, p_2)(p) = \mathrm{in}_2(n + \textsc{bind}(t, p'\ell \to p_2p))$, so $t_1(p) = \mathrm{in}_2(n + \textsc{bind}(t, p'\ell \to p_2p))$. Subcase 1, $p' < p_1$: Then $\textsc{bind}(t, p'\ell \to p_2p) \geq \textsc{bind}(t, p_1 \to p_2p)$. By $(*)$, $\textsc{bind}(t, p'\ell \to p_2p) \geq \textsc{bind}(t, p_1 \to p_1p)$, so by Lemma 3.6, $\textsc{bind}(t, p'\ell \to p_2p) \geq \textsc{bind}(\textsc{subtree}(t, p_1), p)$. By definition of shift, $\textsc{subtree}(t, p_1)(p) = \mathrm{in}_2(n + \textsc{bind}(t, p'\ell \to p_2p) - \textsc{bind}(t, p_1 \to p_2))$. By definition and $(*)$, $t(p_1p) = \mathrm{in}_2(n+\textsc{bind}(t, p'\ell \to p_1p))$. By Lemma 3.4, $\textsc{varof}(t, p_1p) = \mathsf{bound}(p', \ell, n)$ and $p'\ R\ p'$, as required. Subcase 2, $p_1 \leq p' < p_2$: By Lemma 3.4, $\textsc{varof}(\textsc{subtree}(t, p_2), p) = \mathsf{free}(n + \textsc{bind}(t, p'\ell \to p_2p) - \textsc{bind}(\textsc{subtree}(t, p_2), p))$. By Lemma 3.6, the latter is $\mathsf{free}(n+\textsc{bind}(t, p'\ell \to p_2p) - \textsc{bind}(t, p_2 \to p_2p))$, which by definition is $\mathsf{free}(n+\textsc{bind}(t, p'\ell \to p_2))$. So $n+\textsc{bind}(t, p'\ell \to$

$p_2) \in \textsc{ftv}(\textsc{subtree}(t, p_2))$. By Lemma 3.5, $\textsc{shift}(\textsc{subtree}(t, p_1), \textsc{bind}(t, p_1 \to p_2))$ does not have any free variables less than $\textsc{bind}(t, p_1 \to p_2)$, and $n+\textsc{bind}(t, p'\ell \to p_2) < \textsc{bind}(t, p_1 \to p_2)$, a contradiction. Subcase 3, $p' = p_2p''$: By $(*)$, $\textsc{bind}(t, p'\ell \to p_2p) = \textsc{bind}(t, p_1p'' \to p_1p)$. By definition, $n + \textsc{bind}(t, p_1p'' \to p_1p) < \textsc{bind}(t, p_1 \to p_1p)$. By Lemma 3.6, the latter equals $\textsc{bind}(\textsc{subtree}(t, p_1), p)$, so $n + \textsc{bind}(t, p'\ell \to p_2p) < \textsc{bind}(\textsc{subtree}(t_1, p), p)$. By definition of shift, $\textsc{subtree}(t, p_1)(p) = \text{in}_2(n+\textsc{bind}(p_2p''\ell \to p_2p,))$. By definition and $(*)$, $t(p_1p) = \text{in}_2(n+\textsc{bind}(t, p_1p''\ell \to p_1p))$. By Lemma 3.4, $\textsc{varof}(t, p_1p) = \mathsf{bound}(p_1p'', \ell, n)$ and $p_1p''\ R\ p_2p''$, as required. Thus R is an equivalence of t's subtrees. To show that R is nonoverlapping, assume that it is not. Then there is a minimal p such that $\textsc{varof}(t, p_2pp_3) = \mathsf{bound}(p_1p, \ell, n)$ and $p_3 > \epsilon$ for some p_3, ℓ, and n. By Lemma 3.4, $t(p_1p) = \text{in}_1(nl)$, $0 \leq n < \textsc{bind}(nl, \ell)$, and $t(p_2pp_3) = \text{in}_2(n+\textsc{bind}(t, p_1p\ell \to p_2pp_3))$. If $p_1p = p_2p'$ for some p' then by reasoning similar to case 3 subcase 3 above, $\textsc{varof}(t, p_2p'p_3) = \textsc{varof}(t, p_1pp_3) = \mathsf{bound}(p_1p', \ell, n)$ and thus p is not minimal. If $p_1p < p_2$ then by Lemma 3.4, $\textsc{varof}(\textsc{subtree}(t, p_2), pp_3) = \mathsf{free}(n+\textsc{bind}(t, p_1p\ell \to p_2))$. So $n + \textsc{bind}(t, p_1p\ell \to p_2) \in \textsc{ftv}(t_1)$ and $n + \textsc{bind}(t, p_1p\ell \to p_2) < \textsc{bind}(t, p_1 \to p_2)$, a contradiction. Thus R is a nonoverlapping equivalence of t's subtrees, so $R \subseteq \textsc{eqst}(t)$, and, in particular, $p_1\ \textsc{eqst}(R)\ p_2$. □

The next lemma shows that $\textsc{eqst}(t)$ is in fact the largest nonoverlapping equivalence of t's subtrees.

Lemma 3.10 *If $t \in$ Tree then $\textsc{eqst}(t)$ is a nonoverlapping equivalence of t's subtrees.*

Proof: The proof shows that $\textsc{eqst}(t)^+$ is a nonoverlapping equivalence of t's subtrees. Then, $\textsc{eqst}(t)^+$ is contained in $\textsc{eqst}(t)$, which means that $\textsc{eqst}(t)$ equals $\textsc{eqst}(t)^+$, and so is a nonoverlapping equivalence of t's subtrees as required. Since $\textsc{eqst}(t)$ is a union of equivalence relations, $\textsc{eqst}(t)^+$ is an equivalence relation. Let $p_0\ \textsc{eqst}(t)^*\ p_n$ then there exists $n \geq 1$ and $p_1, \ldots, p_{n-1}$ such that $p_{i-1}\ \textsc{eqst}(t)\ p_i$ for $1 \leq i \leq n$. This means there are $R_1, \ldots, R_n$ such that $p_{i-1}\ R_i\ p_i$ and R_i is an equivalence of t's subtrees for $1 \leq i \leq n$. This means that $\textsc{eqstprop}(p_{i-1}, R_i, p_i)$ for $1 \leq i \leq n$. Case 1, $t(p_0) = \text{in}_1(nl)$: Then by a simple induction and $\textsc{eqstprop}(p_{i-1}, R_i, p_i)$, $t(p_i) = \text{in}_1(nl)$ for $0 \leq i \leq n$. In particular $t(p_n) = \text{in}_1(nl)$. Let $\ell \in \textsc{labels}(nl)$. Then by a simple induction and $\textsc{eqstprop}(p_{i-1}, R_i, p_i)$, $p_{i-1}\ell\ R_i\ p_i\ell$. This means that $p_{i-1}\ell\ \textsc{eqst}(t)\ p_i\ell$, which mens that $p_0\ell\ \textsc{eqst}(t)^+\ p_n\ell$, so $\textsc{eqstprop}(p_0, \textsc{eqst}(t)^*, p_n)$, as required. Case 2, $\textsc{varof}(t, p_0) = \mathsf{free}(j)$: Then by a simple induction and $\textsc{eqstprop}(p_{i-1}, R_i, p_i)$, $\textsc{varof}(t, p_i) = \mathsf{free}(j)$ for $0 \leq i \leq n$. In particular $\textsc{varof}(t, p_n) = \mathsf{free}(j)$, so $\textsc{eqstprop}(p_0, \textsc{eqst}(t)^*, p_n)$, as required. Case 3, $\textsc{varof}(t, p_0) = \mathsf{bound}(p'_0, \ell, j)$: Then by induction and $\textsc{eqstprop}(p_{i-1}, R_i, p_i)$, $\textsc{varof}(t, p_i) = \mathsf{bound}(p'_i, \ell, j)$ for $0 \leq i \leq n$ and $p'_{i-1}\ R_i\ p'_i$ for $1 \leq i \leq n$. This means that $p'_{i-1}\ \textsc{eqst}(t)\ p'_i$ for $1 \leq i \leq n$, which means that $p'_0\ \textsc{eqst}(t)^+\ p'_n$. Thus $\textsc{eqstprop}(p_0, \textsc{eqst}(t)^*, p_n)$, as required.

Finally, consider whether $\textsc{eqst}(t)^+$ is nonoverlapping. Assume that $\textsc{eqst}(t)^+$ is not nonoverlapping. Then there exists p_1, ℓ, p_2, p_3, and n such that $\textsc{varof}(t, p_1\ell p_2p_3) = \mathsf{bound}(p_1, \ell, n)$ (1), $p_1\ \textsc{eqst}(t)^+\ p_1\ell p_2$ (2), $p_3 > \epsilon$, and not $p_1\ \textsc{eqst}(t)^+\ p'$ for $p_1\ell p_2 < p' < p_1\ell p_2p_3$ (3). Pick a minimal such p_1. Consider any $p'_3 < p_3$. If $p_1\ \textsc{eqst}(t)^+\ p_1p'_3$ then by (2) and Lemma 3.9, $p_1p'_3\ \textsc{eqst}(t)^+\ p_1\ell p_2p'_3$, so by transitivity, $p_1\ \textsc{eqst}(t)^+\ p_1\ell p_2p'_3$. By (3), this is not possible, so not $p_1\ \textsc{eqst}(t)^+\ p'$ for $p_1 < p' < p_1p_3$ (4). By (2) and Lemma 3.9, $p_1p_3\ \textsc{eqst}(t)^*\ p_1\ell p_2p_3$, so by definition and (1), $\textsc{varof}(t, p_1p_3) = \mathsf{bound}(p'_1, \ell, n)$ for some p'_1 such that $p'_1\ \textsc{eqst}(t)^+\ p_1$. Since p_1 was minimal and (4), it must be that $p'_1 = p_1$, that is, $\textsc{varof}(t, p_1p_3) = \mathsf{bound}(p_1, \ell, n)$ (5).

By (2), there must be $R_0, \ldots, R_m$ and $q_1, \ldots, q_m$ such that R_i is a nonoverlapping equivalence of t's subtrees, $p_1\ R_0\ q_1$ (6), $q_i\ R_i\ q_{i+1}$ for $1 \leq i < m$ (7), and $q_m\ R_m\ p_1 \ell p_2$ (8). By Lemma 3.9, $p_1p_3\ R_0\ q_1p_3$, $q_ip_3\ R_i\ q_{i+1}p_3$ for $1 \leq i < m$, and $q_mp_3\ R_m\ p_1\ell p_2p_3$. By definition, (5), and a simple induction, there must be $q'_1, \ldots, q'_n$ such that $\text{VAROF}(t, q_ip_3) = \mathsf{bound}(q'_i, \ell, n)$, $p_1\ R_0\ q'_1$ (9), $q'_i\ R_i\ q'_{i+1}$ for $1 \leq i < m$ (10), and $q'_m\ R_m\ p_1$ (11). The next step of the proof shows by induction on i that $q_i = q'_i$. For the base case, let $q_0 = q'_0 = p_1$ and note that $q_0\ R_0\ q_1$ (by (6)) and $q'_0\ R_0\ q'_1$ (by (9)), which is all that will be used in the inductive case. For the inductive case, assume that $q_i = q'_i$. This means that $q_i\ R_i\ q_{i+1}$ (by (7)) and $q_i = q'_i\ R_i\ q'_{i+1}$ (by (10)), so $q_{i+1}\ R_i\ q'_{i+1}$ (12). Note that $q'_{i+1} < q_{i+1}p_3$ because of the definition of variable of. If $q'_{i+1} < q_{i+1}$ then R_i would not be nonoverlapping ($q'_{i+1} < q_{i+1} < q_{i+1}p_3$, $\text{VAROF}(t, q_{i+1}p_+3) = \mathsf{bound}(q'_{i+1}, \ell, n)$, and $q'_{i+1}\ R_i\ q_{i+1}$). If $q'_{i+1} = q_{i+1}p'_3$ for some $p'_3 < p_3$ then by Lemma 3.9 and (7), $q_ip'_3\ R_i\ q_{i+1}p'_3$. So $q_ip'_3\ R_i\ q'_{i+1}$ and $q_i = q'_i\ R_i\ q'_{i+1}$, which means that $q_i\ R_i\ q_ip'_3$. A simple induction extends this to $q_0\ R_0\ q_0p'_3$, that is, $p_0\ R_0\ p_0p'_3$, but this contradicts (4). So it must be the case that $q'_{i+1} = q_{i+1}$. Thus by induction, $q'_n = q_n$. So $q_n\ R_n\ p_1\ell p_2$ (by (8)) and $q_q = q'_n\ R_n\ p_1$ (by (11)), which means that $p_1\ R_n\ p_1\ell p_2$. But this contradicts that R_n is nonoverlapping. Thus the original assumption that $\text{EQST}(t)^+$ was not nonoverlapping is false, that is, $\text{EQST}(t)^+$ is nonoverlapping, as required. □

Corollary 3.11 *If $t \in \mathit{Tree}$ then: $t \in \mathit{RegTree}$ if and only if $\text{EQST}(t)$ has a finite number of equivalence classes.*

Proof: Immediate from Lemma 3.10 and definition. □

Several constructions produce regular trees, this is needed to show that terms map to regular trees.

Lemma 3.12

- $\mathsf{var}(n) \in \mathit{RegTree}$
- *If $nl \in NL$ and if $t_\ell \in \mathit{RegTree}$ for $\ell \in \text{LABELS}(nl)$ then $\mathsf{nl}(nl, \ell = t_\ell) \in \mathit{RegTree}$.*
- $\text{SHIFT}(t, n) \in \mathit{RegTree}$ *if and only if* $t \in \mathit{RegTree}$
- *If P is an incomparable set of paths, $\epsilon \notin P$, $t \in \mathit{RegTree}$, $\text{VAROF}(t, p) = \mathsf{free}(n)$ for $p \in P$, and $\text{VAROF}(t, p) \neq \mathsf{free}(n)$ for $p \in \text{dom}(t) - P$ then $\text{fix}(\lambda t'.t\{P := t'\}) \in \mathit{RegTree}$.*

Proof: For the first item, $\text{EQST}(t) = \{(\epsilon, \epsilon)\}$. For the second item, if R_ℓ is a nonoverlapping equivalence of t_ℓ's subtrees with a finite number of equivalence classes then $\{(\epsilon, \epsilon)\} \cup_\ell \{(\ell p_1, \ell p_2) \mid (p_1, p_2) \in R_\ell\}$ is a nonoverlapping equivalence of $\mathsf{nl}(nl, \ell = t_\ell)$'s subtrees and has a finite number of equivalence classes. For the third item, it is easy to show that R is a nonoverlapping equivalence of t's subtrees if and only if it is a nonoverlapping equivalence of $\text{SHIFT}(t, n)$'s subtrees.

For the last item, let $f = \lambda t'.t\{P := t'\}$ and $t_f = \text{fix}(f)$ (note that f is contractive since $\epsilon \notin P$). Let R be a nonoverlapping equivalence of t's subtrees that has a finite number of equivalence classes. Since only paths in P are the free variable n and the definition of equivalence of subtrees, $p\ R\ p'$ and $p \in P$ implies that $p' \in P$ $(**)$. Let $P' = \{p \mid p \in \mathrm{dom}(t) \wedge \forall p' \in P : p' \not\leq p\}$ and $R' = R \cap P' \times P'$. Since R is an equivalence relation on $\mathrm{dom}(t)$, R' is an equivalence relation on P'. Let $S = \{(p_1 p_1', p_2 p_2') \mid p_1 \in P^* \wedge p_2 \in P^* \wedge (p_1', p_2') \in R'\}$; clearly S is an equivalence on $P^* P'$. By the fixed point property and induction $t_f = f^i(t_f)$ for any $i \in \mathbb{N}$. Thus $p \in \mathrm{dom}(t_f)$ if and only if $p \in \mathrm{dom}(f^{|p|+1}(t_f))$, which is if and only if $p \in P^* P'$. Thus S is an equivalence relation on $\mathrm{dom}(t_f)$.

Let $p_1 p_1'\ S\ p_2 p_2'$ where $p_1, p_2 \in P^*$ and $p_1', p_2' \in P'$. Since $t_f = f^i(t_f)$ for all i and by induction, $\textsc{subtree}(t_f, ()p_i) = \textsc{shift}(t_f, \textsc{bind}(t_f, p_i))$. Since $t_f(p_i') = t(p_i')$,

$$t_f(p_i p_i') = \textsc{shift}(t, \textsc{bind}(t_f, p_i))(p_i') \qquad (*)$$

Case 1, $t(p_1') = \mathrm{in}_1(nl)$: Then since R is an equivalence of t's subtrees, $t(p_2') = \mathrm{in}_1(nl)$ and $p_1'\ell\ R\ p_2'\ell$ for all $\ell \in \textsc{labels}(nl)$. Thus $t_f(p_i p_i') = \mathrm{in}_1(nl)$. If $p_1'\ell \in P'$ then by $(**)$, $p_2'\ell \in P'$ so $p_1 p_1'\ell\ S\ p_2 p_2'\ell$, as required for S to be an equivalence of t_f's subtrees. If $p_1'\ell \notin P'$ then $p_1'\ell \in P$ and by $(**)$, $p_2'\ell \in P$. Thus $p_i p_i'\ell \in P^*$ and since $(\epsilon, \epsilon) \in R'$, $p_1 p_1'\ell\ S\ p_2 p_2'\ell$, as required for S to be an equivalence of t_f's subtrees. Case 2, $\textsc{varof}(t, p_1') = \mathsf{free}(n)$: Then since R is an equivalence of t's subtrees, $\textsc{varof}(t, p_2') = \mathsf{free}(n)$. By $(*)$ and Lemma 3.4, $\textsc{varof}(t_f, p_i p_i') = \mathsf{free}(n)$, as required for S to an equivalence of t_f's subtrees. Case 3, $\textsc{varof}(t, p_1') = \mathsf{bound}(p_1'', \ell, n)$: Then since R is an equivalence of t's subtrees, $\textsc{varof}(t, p_2') = \mathsf{bound}(p_2'', \ell, n)$ and $p_1''\ R'\ p_2''$ (note that $p_i'' \in P'$ because they are prefixes of p_i' respectively). By $(*)$ and Lemma 3.4, $\textsc{varof}(t_f, p_i p_i') = \mathsf{bound}(p_i p_i'', \ell, n)$ and clearly $p_1 p_1''\ S\ p_2 p_2''$, as required for S to be an equivalence of t_f's subtrees. Thus S is an equivalence of t_f's subtrees.

Let $\textsc{varof}(t, p_1 p_2) = \mathsf{bound}(p', \ell, n)$ where $p_1 \in P^*$ and $p_2 \in P'$. First note that $p' \not< p_1$ because $\textsc{subtree}(t_f, p_1)$ is of the form $\textsc{shift}(t', \textsc{bind}(t_f, p_1))$ fro some t'. Thus $p' = p_1 p_2'$. By $(*)$ and Lemma 3.4, $\textsc{varof}(t, p_2) = \mathsf{bound}(p_2', \ell, n)$. Since R is nonoverlapping, $(p_2', p'') \notin R$ for $p_2' < p'' < p_2$. Thus $(p_1 p_2', p'') \notin S$ for $p_1 p_2' < p'' < p_1 p_2$ as required for S to be nonoverlapping. The number of equivalence classes of S is equal to the number of equivalence classes of R' which is less than the number of equivalence classes of R. Thus S is a nonoverlapping equivalence of t_f's subtrees with a finite number of equivalence classes, as required. $\square$

4 Terms

The second-order term language with recursive quantifiers is:

$$\tau ::= \alpha \mid nl(\ell = \vec{\alpha}_\ell.\tau_\ell)_{\ell \in \mathrm{LABELS}(nl)} \mid \mathsf{rec}\ \alpha.\tau$$

where α ranges over the set of term variables $TVar$ and $\vec{\alpha}$ is a sequence of term variables. Terms are considered equal up to alpha conversion.

Not all phrases matching the above grammar are considered terms, but only those that satisfy two further constraints. In terms of the from $nl(\ell = \vec{\alpha}_\ell.\tau_\ell)$ it must be that $|\vec{\alpha}_\ell| = \textsc{bind}(nl, \ell)$. In terms of the form $\mathsf{rec}\ \alpha.\tau$ it must be that $\tau \downarrow \alpha$, where the latter means that τ is syntactically contractive in α and is defined as:

$$\begin{array}{lll} \beta \downarrow \alpha & \Leftrightarrow & \alpha \neq \beta \\ nl(\ell = \vec{\alpha_\ell}.\tau_\ell) \downarrow \alpha & & \\ \mathsf{rec}\ \beta.\tau \downarrow \alpha & \Leftrightarrow & \alpha = \beta \vee \tau \downarrow \alpha \end{array}$$

Intuitively, $\tau \downarrow \alpha$ if mapping α to τ is not equivalent to the identity mapping, for which any term is a fixed point. Instead mapping α to τ produces a term whose outer most constructor is independent of α, and can have only one fixed point.

4.1 Terms to Trees

The interpretation of terms as trees depends upon the interpretation of its free variables.

Definition 4.1 *The environments, ranged over by metavariable η, are $Env = TVar \to Tree$. If $\eta \in Env$ then the shift of η by the vector of term variables $\alpha_0 \cdots \alpha_{n-1}$ is defined as:*

$$\textsc{shift}(\eta, \alpha_0, \ldots, \alpha_{n-1}) = \lambda\beta. \begin{cases} \mathsf{var}(i) & \beta = \alpha_i \\ \textsc{shift}(\eta(\beta), n) & \beta \notin \{\alpha_0, \ldots, \alpha_{n-1}\} \end{cases}$$

The shifting operation on environments has some basic properties that are needed in later proofs.

Lemma 4.1

- *If $\eta \in Env$ then $\textsc{shift}(\eta, \vec{\alpha}) \in Env$.*
- *If $\eta \in TVar \to RegTree$ then $\textsc{shift}(\eta, \vec{\alpha}) \in TVar \to RegTree$.*
- *If α is distinct from α_i then $\textsc{shift}(\eta\{\alpha \mapsto t\}, \alpha_1 \cdots \alpha_n) = \textsc{shift}(\eta, \alpha_1 \cdots \alpha_n)\{\alpha \mapsto \textsc{shift}(t, n)\}$.*
- $\textsc{shift}(\textsc{shift}(\eta, \vec{\beta}), \vec{\alpha}) = \textsc{shift}(\eta, \vec{\alpha}\vec{\beta})$

Proof: The results follow straightforwardly from the definitions and (in the case of the second result) Lemma 3.12. □

With these preliminaries, the interpretation of a term as a tree is straightforward.

Definition 4.2 *The meaning of a term τ in an environment η, $[\![\tau]\!]_\eta$ is defined as:*

$$\begin{array}{lll} [\![\alpha]\!]_\eta & = & \eta(\alpha) \\ [\![nl(\ell = \vec{\alpha_\ell}.\tau_\ell)]\!]_\eta & = & \mathsf{nl}(nl, \ell = [\![\tau_\ell]\!]_{\textsc{shift}(\eta, \vec{\alpha_\ell})}) \\ [\![\mathsf{rec}\ \alpha.\tau]\!]_\eta & = & \mathrm{fix}([\![\alpha.\tau]\!]_\eta) \\ [\![\alpha.\tau]\!]_\eta & = & \lambda t.[\![\tau]\!]_{\eta\{\alpha \mapsto t\}} \end{array}$$

To show that this definition uniquely defines a tree, a small technical result is needed.

Lemma 4.2 *If A is an ultrametric space and $f : A \xrightarrow{c} A \xrightarrow{1/2} A$ then $\lambda x.\mathrm{fix}(f(x))$ is c-Lipschitz.*

Proof:[4] Let $x_1, x_2 \in A$. Let $m = d(x_1, x_2)$, $f_1 = f(x_1)$, $f_2 = f(x_2)$, and $y = \mathrm{fix}(f_1)$. Since f is c-Lipschitz then $d(f_1, f_2) \leq cm$. Thus $d(y, f_2(y)) = d(f_1(y), f_2(y)) \leq cm$. Since f_2 is 1/2-Lipschitz, $d(f_2(y), f_2^2(y)) \leq 1/2d(y, f_2(y))$. Since d is an ultrametric, $d(y, f_2^2(y)) = d(y, f_2(y))$. By induction, $d(y, f_2^i(y)) = d(y, f_2(y))$. Since d is continuous, $d(y, \mathrm{fix}(f_2)) = d(y, f_2(y))$. Thus $d(\mathrm{fix}(f_1), \mathrm{fix}(f_2)) \leq cm$, as required. □

It is easy to show that a term's interpretation is a uniquely defined tree, and the proof also shows that syntactically contractive term and variable pairs define contractive maps.

Theorem 4.3 *If $\eta \in Env$ then $[\![\tau]\!]_\eta \in Tree$, $[\![\alpha.\tau]\!]_\eta$ is nonexpansive, and $[\![\alpha.\tau]\!]_\eta$ is contractive if $\tau \downarrow \alpha$.*

Proof: The proof is by simultaneous induction on τ. Consider the various cases:

$\tau = \beta$**:** In this case $[\![\tau]\!]_\eta = \eta(\beta)$, which is in *Tree* since $\eta \in Env$. Now $[\![\alpha.\tau]\!]_\eta$ is either $\lambda t.t$ if $\alpha = \beta$ or is $\lambda t.\eta(\beta)$ if $\alpha \neq \beta$. In the first case $\lambda t.t$ is 1-Lipschitz and $\neg(\tau \downarrow \alpha)$. In the second case $\lambda t.\eta(\beta)$ is 0-contractive.

$\tau = nl(\ell = \vec{\alpha_\ell}.\tau_\ell)$**:** Without loss of generality assume that α is not in $\vec{\alpha_\ell}$. By Lemma 4.1, $\textsc{shift}(\eta, \vec{\alpha_\ell}) \in Env$, so by the induction hypothesis, $[\![\tau_\ell]\!]_{\mathrm{SHIFT}(\eta,\vec{\alpha_\ell})} \in Tree$. By Lemma 3.2, $[\![\tau]\!]_\eta \in Tree$. By the induction hypothesis, $[\![\alpha.\tau_\ell]\!]_{\mathrm{SHIFT}(\eta,\vec{\alpha_\ell})}$ is nonexpansive and by Lemma 3.5, $\lambda t.\textsc{shift}(t, \textsc{bind}(nl, \ell))$ preserves distances. So

$$\lambda t.[\![\tau_\ell]\!]_{\mathrm{SHIFT}(\eta,\vec{\alpha_\ell})\{\alpha\mapsto\mathrm{SHIFT}(t,\mathrm{BIND}(nl,\ell))\}}$$

is nonexpansive. By Lemma 4.1,

$$[\![\tau_\ell]\!]_{\mathrm{SHIFT}(\eta,\vec{\alpha_\ell})\{\alpha\mapsto\mathrm{SHIFT}(t,\mathrm{BIND}(nl,\ell))\}} = [\![\tau_\ell]\!]_{\mathrm{SHIFT}(\eta\{\alpha\mapsto t\},\vec{\alpha_\ell})}$$

Thus, $\lambda t.\mathsf{nl}(nl, \ell = [\![\tau_\ell]\!]_{\mathrm{SHIFT}(\eta\{\alpha\mapsto t\},\vec{\alpha_\ell})})$ is 1/2-contractive. Hence, $[\![\alpha.\tau]\!]_\eta$ is contractive.

$\tau = \mathsf{rec}\ \beta.\sigma$**:** If $t \in Tree$ then $\eta\{\beta \mapsto t\} \in Env$. By the induction hypothesis, $[\![\sigma]\!]_{\eta\{\beta\mapsto t\}} \in Tree$ thus $[\![\beta.\sigma]\!]_\eta$ maps *Tree* to *Tree*. By syntactic constrainst, $\sigma \downarrow \beta$, so by the induction hypothesis, $[\![\beta.\sigma]\!]_\eta$ is contractive. Since *Tree* is a complete ultrametric space, $[\![\beta.\sigma]\!]_\eta$ has a unique fixed point in *Tree*, so $[\![\tau]\!]_\eta \in Tree$. There are two cases for $[\![\alpha.\tau]\!]_\eta$. Case 1, $\alpha = \beta$: In this case $[\![\alpha.\tau]\!]_\eta$ is a constant function and so is 0-contractive. Case 2, $\alpha \neq \beta$: In this case $\tau \downarrow \alpha$ exactly when $\sigma \downarrow \alpha$. To show that $[\![\alpha.\tau]\!]_\eta$ is nonexpansive and/or contractive, it suffices by Lemma 4.2 to show that $\lambda t.\lambda t'.[\![\sigma]\!]_{\eta\{\alpha\mapsto t\}\{\beta\mapsto t'\}}$ is nonexpansive and/or contractive. Expanding the definitions this means showing that $d([\![\sigma]\!]_{\eta\{\alpha\mapsto t_1\}\{\beta\mapsto t'\}}, [\![\sigma]\!]_{\eta\{\alpha\mapsto t_2\}\{\beta\mapsto t'\}}) \leq cd(t_1, t_2)$ for all t', t_1, and t_2 and for $c \leq 1$ ($c < 1$ for contractive) $(*)$. Since $\eta\{\alpha \mapsto t_i\}\{\beta \mapsto t'\} = \eta\{\beta \mapsto t'\}\{\alpha \mapsto t_i\}$, $(*)$ reduces to showing that $[\![\alpha.\sigma]\!]_{\eta\{\beta\mapsto t'\}}$ is nonexpansive and/or contractive for all t'. This holds by the induction hypothesis.

[4]The proof of Lemma 4.2 is due to Dexter Kozen.

□

4.2 Properties of the Interpretation

This section shows a bunch of different properties of the interpretation of terms as trees that are useful in proving other results.

First, the interpretation of a term is dependent upon the environment only on the term's free variables.

Lemma 4.4 *If $\eta_1(\alpha) = \eta_2(\alpha)$ for all $\alpha \in \textsc{ftv}(\tau)$ then $[\![\tau]\!]_{\eta_1} = [\![\tau]\!]_{\eta_2}$.*

Proof: The result follows by a straightforward induction on the structure of τ. □

The occurances of variables in a term correspond in a certain way to paths in the interpretation of that term. This correspondance is useful in stating several of results.

Definition 4.3

$$
\begin{array}{lcl}
\textsc{pathsvar}(\alpha,\alpha) & = & \{\epsilon\} \\
\textsc{pathsvar}(\beta,\alpha) & = & \emptyset \quad \textit{where } \alpha \neq \beta \\
\textsc{pathsvar}(nl(\ell = \vec{\alpha_\ell}.\tau_\ell),\alpha) & = & \{\ell p \mid p \in \textsc{pathsvar}(\tau_\ell,\alpha)\} \quad \textit{where } \alpha \notin \vec{\alpha_\ell} \\
\textsc{pathsvar}(\mathsf{rec}\ \beta.\tau,\alpha) & = & \textsc{pathsvar}(\tau,\beta)^*\textsc{pathsvar}(\tau,\alpha) \quad \textit{where } \alpha \neq \beta
\end{array}
$$

The paths of a variable in a term satisfy some basic properties.

Lemma 4.5

- *The set $\textsc{pathsvar}(\tau,\alpha)$ is incomparable.*
- *If $\alpha \neq \beta$ then $\textsc{pathsvar}(\tau,\alpha) \cap \textsc{pathsvar}(\tau,\beta) = \emptyset$.*
- *If $\tau \downarrow \alpha$ then $\epsilon \notin \textsc{pathsvar}(\tau,\alpha)$.*

Proof: The results follow straightforwardly by induction and the definitions. □

The next result shows that substitution in an environment is equivalent to a substitution on the interpretation. It requires a small technical result, which is presented next.

Lemma 4.6 *If $t, t_2 \in Tree$, $\epsilon \notin P_1$, $P_1 \cup P_2 \subseteq \mathrm{dom}(t)$, $P_1 \cup P_2$ is incomparable, and $P_1 \cap P_2 = \emptyset$ then $\mathrm{fix}(\lambda t_1.t\{P_1 := t_1\}\{P_2 := t_2\}) = \mathrm{fix}(\lambda t_1.t\{P_1 := t_1\})\{P_1^* P_2 := t_2\}$.*

Proof: Note that $P_1 \cup P_2$ and $P_1^* P_2$ are incomparable.

$$\begin{array}{ll}
 & (\lambda t_1.t\{P_1 := t_1\}\{P_2 := t_2\})(\text{fix}(\lambda t_1.t\{P_1 := t_1\})\{P_1^* P_2 := t_2\}) \\
= & \quad \langle \text{Definition} \rangle \\
 & t\{P_1 := \text{fix}(\lambda t_1.t\{P_1 := t_1\})\{P_1^* P_2 := t_2\}\}\{P_2 := t_2\} \\
= & \quad \langle \text{Lemma 3.8} \rangle \\
 & t\{P_1 := \text{fix}(\lambda t_1.t\{P_1 := t_1\})\}\{P_1 P_1^* P_2 := t_2\}\{P_2 := t_2\} \\
= & \quad \langle \text{Lemma 3.8} \rangle \\
 & t\{P_1 := \text{fix}(\lambda t_1.t\{P_1 := t_1\})\}\{P_1^* P_2 := t_2\} \\
= & \quad \langle \text{Fixed point properties} \rangle \\
 & \text{fix}(\lambda t_1.t\{P_1 := t_1\})\{P_1^* P_2 := t_2\}
\end{array}$$

Clearly $\lambda t_1.t\{P_1 := t_1, P_2 := t_2\}$ is contractive because $\epsilon \notin P_1$. Thus the result follows because fixed points of contractive functions are unique. □

Lemma 4.7 $[\![\tau]\!]_{\eta\{\alpha \mapsto t\}} = [\![\tau]\!]_\eta\{\textsc{pathsvar}(\tau, \alpha) := t\}$

Proof: The proof is by induction on τ, consider the cases for τ:

$\tau = \alpha$: In this case $[\![\tau]\!]_{\eta\{\alpha \mapsto t\}} = t$, $\textsc{pathsvar}(\tau, \alpha) = \{\epsilon\}$, and $[\![\tau]\!]_\eta\{\textsc{pathsvar}(\tau, \alpha)t := t\} = t$, as required.

$\tau = \beta \neq \alpha$: In this case $[\![\tau]\!]_{\eta\{\alpha \mapsto t\}} = \eta(\beta)$, $\textsc{pathsvar}(\tau, \alpha) = \emptyset$, $[\![\tau]\!]_\eta\{\textsc{pathsvar}(\tau, \alpha)t := t\} = [\![\tau]\!]_\eta = \eta(\beta)$, as required.

$\tau = nl(\ell = \vec{\alpha_\ell}.\tau_1)$: Without loss of generality assume that $\alpha \notin \vec{\alpha_\ell}$. Then:

$$\begin{array}{ll}
 & [\![\tau]\!]_{\eta\{\alpha \mapsto t\}} = \\
= & \quad \langle \text{Definition} \rangle \\
 & \mathsf{nl}(nl, \ell = [\![\tau_\ell]\!]_{\text{SHIFT}(\eta\{\alpha \mapsto t\}, \vec{\alpha_\ell})}) \\
= & \quad \langle \text{Lemma 4.1} \rangle \\
 & \mathsf{nl}(nl, \ell = [\![\tau_\ell]\!]_{\text{SHIFT}(\eta, \vec{\alpha_\ell})\{\alpha \mapsto \text{SHIFT}(t, |\vec{\alpha_\ell}|)\}}) \\
= & \quad \langle \text{Induction Hypothesis} \rangle \\
 & \mathsf{nl}(nl, \ell = [\![\tau_\ell]\!]_{\text{SHIFT}(\eta, \vec{\alpha_\ell})}\{\textsc{pathsvar}(\tau_\ell, \alpha) := \textsc{shift}(t, |\vec{\alpha_\ell}|)\}) \\
= & \quad \langle \text{Lemma 3.8} \rangle \\
 & \mathsf{nl}(nl, \ell = [\![\tau_\ell]\!]_{\text{SHIFT}(\eta, \vec{\alpha_\ell})})\{\{\ell p \mid p \in \textsc{pathsvar}(\tau_\ell, \alpha)\} := t\} \\
= & \quad \langle \text{Definition} \rangle \\
 & [\![\tau]\!]_\eta\{\textsc{pathsvar}(\tau, \alpha) := t\}
\end{array}$$

$\tau = \mathsf{rec}\ \beta.\sigma$**:** Without loss of generality assume that $\alpha \neq \beta$, then:

$$
\begin{array}{ll}
 & [\![\tau]\!]_{\eta\{\alpha\mapsto t\}} = \\
= & \quad \langle\text{Definition}\rangle \\
 & \text{fix}(\lambda t'.[\![\sigma]\!]_{\eta\{\alpha\mapsto t\}\{\beta\mapsto t'\}}) \\
= & \quad \langle\text{Induction Hypothesis twice}\rangle \\
 & \text{fix}(\lambda t'.[\![\sigma]\!]_\eta\{\text{PATHSVAR}(\sigma,\beta) := t'\}\{\text{PATHSVAR}(\sigma,\alpha) := t\}) \\
= & \quad \langle\text{Lemmas 4.6 and 4.5}\rangle \\
 & \text{fix}(\lambda t'.[\![\sigma]\!]_\eta\{\text{PATHSVAR}(\sigma,\beta) := t'\})\{\text{PATHSVAR}(\sigma,\beta)^*\text{PATHSVAR}(\sigma,\alpha) := t\} \\
= & \quad \langle\text{Induction Hypothesis}\rangle \\
 & \text{fix}(\lambda t'.[\![\sigma]\!]_{\eta\{\beta\mapsto t'\}})\{\text{PATHSVAR}(\sigma,\beta)^*\text{PATHSVAR}(\sigma,\alpha) := t\} \\
= & \quad \langle\text{Definition}\rangle \\
 & [\![\tau]\!]_\eta\{\text{PATHSVAR}(\tau,\alpha) := t\}
\end{array}
$$

□

The results so far can be used to show that terms map to regular trees. Also, all regular trees are also expressible as terms; this is proven in the section on completeness.

Theorem 4.8 *If $\eta \in \mathit{TVar} \to \mathit{RegTree}$ then $[\![\tau]\!]_\eta \in \mathit{RegTree}$.*

Proof: The proof is by induction on the structure of τ. Consider the cases for τ:

$\tau = \alpha$**:** In this case, $[\![\tau]\!]_\eta = \eta(\alpha)$, which is a regular tree since $\eta \in \mathit{TVar} \to \mathit{RegTree}$.

$\tau = nl(\ell = \vec{\alpha_\ell}.\tau_\ell)$**:** By Lemma 4.1, $\text{SHIFT}(\eta, \vec{\alpha_\ell}) \in \mathit{TVar} \to \mathit{RegTree}$. By the induction hypothesis $[\![\tau_\ell]\!]_{\text{SHIFT}(\eta,\vec{\alpha_\ell})} \in \mathit{RegTree}$. By Lemma 3.12, $[\![\tau]\!]_\eta \in \mathit{RegTree}$.

$\tau = \mathsf{rec}\ \alpha.\sigma$**:** By the induction hypothesis, $[\![\sigma]\!]_\eta \in \mathit{RegTree}$. Since each free variable requires its own equivalence class, $\text{FTV}([\![\sigma]\!]_\eta)$ is finite. Let n be in $\mathbb{N} - \text{FTV}([\![\sigma]\!]_\eta)$ and $\eta' = \eta\{\alpha \mapsto \mathsf{var}(n)\}$. Since $\alpha \notin \text{FTV}(\tau)$, by Lemma 4.4, $[\![\tau]\!]_\eta = [\![\tau]\!]_{\eta'}$. By Lemma 3.12, $\eta' \in \mathit{TVar} \to \mathit{RegTree}$. By the induction hypothesis, $[\![\sigma]\!]_{\eta'} \in \mathit{RegTree}$. By Lemma 4.7, $[\![\sigma]\!]_{\eta'} = [\![\sigma]\!]_\eta\{\text{PATHS}(\sigma,\alpha) := \mathsf{var}(n)\}$. Thus $\text{VAROF}([\![\sigma]\!]_{\eta'}, p) = \mathsf{free}(n)$ for $p \in \text{PATHS}(\sigma,\alpha)$, and $\text{VAROF}([\![\sigma]\!]_{\eta'}, p) \neq \mathsf{free}(n)$ for $p \in \text{dom}([\![\sigma]\!]_{\eta'}) - \text{PATHS}(\sigma,\alpha)$. By Lemma 4.5, the set $\text{PATHSVAR}(\sigma,\alpha)$ is incomparable and $\epsilon \notin \text{PATHSVAR}(\sigma,\alpha)$. Thus by Lemma 3.12, $\text{fix}(\lambda t.[\![\sigma]\!]_{\eta'}\{\text{PATHSVAR}(\sigma,\alpha) := t\}) \in \mathit{RegTree}$. By Lemma 4.7, $\text{fix}(\lambda t.[\![\sigma]\!]_{\eta'\{\alpha\mapsto t\}}) \in \mathit{RegTree}$. By definition, $[\![\tau]\!]_{\eta'} \in \mathit{RegTree}$, so $[\![\tau]\!]_\eta \in \mathit{RegTree}$.

□

The next result shows that shifting the interpretation of a term is equivalent to shifting the environment with fresh variables.

Lemma 4.9 *If $\vec{\alpha} \cap \text{FTV}(\tau) = \emptyset$ then $[\![\tau]\!]_{\text{SHIFT}(\eta,\vec{\alpha})} = \text{SHIFT}([\![\tau]\!]_\eta, |\vec{\alpha}|)$.*

Proof: The proof shows that if $\vec{\alpha} \cap \text{FTV}(\tau) = \emptyset$, $|\vec{\alpha}| = n$, and $|\vec{\beta}| = m$ then $[\![\tau]\!]_{\text{SHIFT}(\eta,\vec{\beta}\vec{\alpha})} = \text{SHIFT}([\![\tau]\!]_{\text{SHIFT}(\eta,\vec{\beta})}, m, n)$ by induction on τ. Consider the cases for τ:

$\tau = \gamma \in \vec{\beta}$**:** In this case $[\![\tau]\!]_{\text{SHIFT}(\eta,\vec{\beta}\vec{\alpha})} = \mathsf{var}(i)$ for some $i < m$, also $[\![\tau]\!]_{\text{SHIFT}(\eta,\vec{\beta})} = \mathsf{var}(i)$, so $\textsc{shift}([\![\tau]\!]_{\text{SHIFT}(\eta,\vec{\beta})}, m, n) = \mathsf{var}(i)$, as required.

$\tau = \gamma \notin \vec{\beta}$**:** In this case $\tau \notin \vec{\alpha}$, so $[\![\tau]\!]_{\text{SHIFT}(\eta,\vec{\beta}\vec{\alpha})} = \textsc{shift}(\eta(\gamma), n + m)$ and $[\![\tau]\!]_{\text{SHIFT}(\eta,\vec{\beta})} = \textsc{shift}(\eta(\gamma), m)$, so by Lemma 3.5, $\textsc{shift}([\![\tau]\!]_{\text{SHIFT}(\eta,\vec{\beta})}, m, n) = \textsc{shift}(\eta(\gamma), n + m)$, as required.

$\tau = nl(\ell = \vec{\gamma_\ell}.\tau_\ell)$**:** Without loss of generality assume that $\vec{\alpha} \cap \vec{\gamma_\ell} = \emptyset$, then:

$$
\begin{array}{cl}
& [\![\tau]\!]_{\text{SHIFT}(\eta,\vec{\beta}\vec{\alpha})} \\
= & \quad \langle \text{Definition} \rangle \\
& \mathsf{nl}(nl, \ell = [\![\tau_\ell]\!]_{\text{SHIFT}(\text{SHIFT}(\eta,\vec{\beta}\vec{\alpha}),\vec{\gamma_\ell})}) \\
= & \quad \langle \text{Lemma 4.1} \rangle \\
& \mathsf{nl}(nl, \ell = [\![\tau_\ell]\!]_{\text{SHIFT}(\eta,\vec{\gamma_\ell}\vec{\beta}\vec{\alpha})}) \\
= & \quad \langle \text{Induction Hypothesis} \rangle \\
& \mathsf{nl}(nl, \ell = \textsc{shift}([\![\tau_\ell]\!]_{\text{SHIFT}(\eta,\vec{\gamma_\ell}\vec{\beta})}, m + |\vec{\gamma_\ell}|, n)) \\
= & \quad \langle \text{Lemma 3.5} \rangle \\
& \textsc{shift}(\mathsf{nl}(nl, \ell = [\![\tau_\ell]\!]_{\text{SHIFT}(\eta,\vec{\gamma_\ell}\vec{\beta})}), m, n) \\
= & \quad \langle \text{Lemma 4.1} \rangle \\
& \textsc{shift}(\mathsf{nl}(nl, \ell = [\![\tau_\ell]\!]_{\text{SHIFT}(\text{SHIFT}(\eta,\vec{\beta}),\vec{\gamma_\ell})}), m, n) \\
= & \quad \langle \text{Definition} \rangle \\
& \textsc{shift}([\![\tau]\!]_{\text{SHIFT}(\eta,\vec{\beta})}, m, n)
\end{array}
$$

$\tau = \mathsf{rec}\ \gamma.\sigma$**:** Without loss of generality assume that $\gamma \notin \vec{\alpha}$, then:

$$
\begin{array}{cl}
& [\![\tau]\!]_{\text{SHIFT}(\eta,\vec{\beta}\vec{\alpha})} \\
= & \quad \langle \text{Definition} \rangle \\
& \text{fix}(\lambda t.[\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\beta}\vec{\alpha})\{\gamma \mapsto t\}}) \\
= & \quad \langle \text{Lemma 4.7} \rangle \\
& \text{fix}(\lambda t.[\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\beta}\vec{\alpha})}\{\textsc{pathsvar}(\sigma, \gamma) := t\}) \\
= & \quad \langle \text{Induction Hypothesis} \rangle \\
& \text{fix}(\lambda t.\textsc{shift}([\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\beta})}, m, n)\{\textsc{pathsvar}(\sigma, \gamma) := t\}) \\
= & \quad \langle \text{See below} \rangle \\
& \textsc{shift}(\text{fix}(\lambda t.[\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\beta})}\{\textsc{pathsvar}(\sigma, \gamma) := t\}), m, n) \\
= & \quad \langle \text{Lemma 4.7} \rangle \\
& \textsc{shift}(\text{fix}(\lambda t.[\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\beta})\{\gamma := t\}}), m, n) \\
= & \quad \langle \text{Definition} \rangle \\
& \textsc{shift}([\![\tau]\!]_{\text{SHIFT}(\eta,\vec{\beta})}, m, n)
\end{array}
$$

The middle step requires showing that

$$\textsc{shift}(\text{fix}(\lambda t.[\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\beta})}\{\textsc{pathsvar}(\sigma, \gamma) := t\}), m, n)$$

is equal to $\text{fix}(f)$ where $f = \lambda t.\textsc{shift}([\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\beta})}, m, n)\{\textsc{pathsvar}(\sigma, \gamma) := t\}$. This follows if it is a fixed point of f and f is contractive. By syntactic constraints, $\sigma \downarrow \gamma$. By

Lemma 4.5, $\epsilon \notin \text{PATHSVAR}(\sigma,\gamma)$, so f is contractive. Finally:

$$\begin{array}{ll}
& f(\text{SHIFT}(\text{fix}(\lambda t.[\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\beta})}\{\text{PATHSVAR}(\sigma,\gamma) := t\}), m, n)) \\
= & \quad \langle\text{Definition}\rangle \\
& \text{SHIFT}([\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\beta})}, m, n) \\
& \quad \{\text{PATHSVAR}(\sigma,\gamma) := \text{SHIFT}(\text{fix}(\lambda t.[\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\beta})}\{\text{PATHSVAR}(\sigma,\gamma) := t\}), m, n)\} \\
= & \quad \langle\text{Lemma 3.8}\rangle \\
& \text{SHIFT}([\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\beta})}\{\text{PATHSVAR}(\sigma,\gamma) := \\
& \qquad\qquad \text{fix}(\lambda t.[\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\beta})}\{\text{PATHSVAR}(\sigma,\gamma) := t\})\}, m, n) \\
= & \quad \langle\text{Fixed point properties}\rangle \\
& \text{SHIFT}(\text{fix}(\lambda t.[\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\beta})}\{\text{PATHSVAR}(\sigma,\gamma) := t\}), m, n)
\end{array}$$

□

The next result shows that the interpretation of a syntactic substitution is equivalent to a substitution in the environment.

Lemma 4.10 $[\![\tau]\!]_{\eta\{\alpha\mapsto[\![\sigma]\!]_\eta\}} = [\![\tau\{\alpha := \sigma\}]\!]_\eta$ *and* $[\![\alpha.\tau]\!]_\eta([\![\sigma]\!]_\eta) = [\![\tau\{\alpha := \sigma\}]\!]_\eta$

Proof: The second item follows from the first by definition. The proof of the first item is by induction on the structure of τ. Consider the cases for τ:

$\tau = \beta \neq \alpha$: In this case $[\![\tau]\!]_{\eta\{\alpha\mapsto[\![\sigma]\!]_\eta\}} = \eta\{\alpha \mapsto [\![\sigma]\!]_\eta\}(\beta) = \eta(\beta) = [\![\tau]\!]_\eta = [\![\tau\{\alpha := \sigma\}]\!]_\eta$.

$\tau = \alpha$: In this case $[\![\tau]\!]_{\eta\{\alpha\mapsto[\![\sigma]\!]_\eta\}} = \eta\{\alpha \mapsto [\![\sigma]\!]_\eta\}(\alpha) = [\![\sigma]\!]_\eta = [\![\tau\{\alpha := \sigma\}]\!]_\eta$.

$\tau = nl(\ell = \vec{\alpha_\ell}.\tau_\ell)$: Without loss of generality assume that $\vec{\alpha_\ell} \cap \text{FTV}(\sigma) = \emptyset$ for $\ell \in \text{LABELS}(nl)$. Then:

$$\begin{array}{ll}
& [\![\tau]\!]_{\eta\{\alpha\mapsto[\![\sigma]\!]_\eta\}} \\
= & \quad \langle\text{Definition}\rangle \\
& \mathsf{nl}(nl, \ell = [\![\tau_\ell]\!]_{\text{SHIFT}(\eta\{\alpha\mapsto[\![\sigma]\!]_\eta\},\vec{\alpha_\ell})}) \\
= & \quad \langle\text{Lemma 4.1}\rangle \\
& \mathsf{nl}(nl, \ell = [\![\tau_\ell]\!]_{\text{SHIFT}(\eta,\vec{\alpha_\ell})\{\alpha\mapsto\text{SHIFT}([\![\sigma]\!]_\eta,|\vec{\alpha_\ell}|)\}}) \\
= & \quad \langle\text{Lemma 4.9}\rangle \\
& \mathsf{nl}(nl, \ell = [\![\tau_\ell]\!]_{\text{SHIFT}(\eta,\vec{\alpha_\ell})\{\alpha\mapsto[\![\sigma]\!]_{\text{SHIFT}(\eta,\vec{\alpha_\ell})}\}}) \\
= & \quad \langle\text{Induction hypothesis}\rangle \\
& \mathsf{nl}(nl, \ell = [\![\tau_\ell\{\alpha := \sigma\}]\!]_{\text{SHIFT}(\eta,\vec{\alpha_\ell})}) \\
= & \quad \langle\text{Definition}\rangle \\
& [\![nl(\ell = \vec{\alpha_\ell}.\tau_\ell\{\alpha := \sigma\})]\!]_\eta \\
= & \\
& [\![\tau\{\alpha := \sigma\}]\!]_\eta
\end{array}$$

$\tau = \mathsf{rec}\ \beta.\tau'$**:** Without loss of generality assume that $\alpha \neq \beta$ and $\beta \notin \text{FTV}(\sigma)$. Then:

$$
\begin{array}{ll}
 & [\![\tau]\!]_{\eta\{\alpha\mapsto[\![\sigma]\!]_\eta\}} \\
= & \quad \langle\text{Definition}\rangle \\
 & \text{fix}(\lambda t.[\![\tau']\!]_{\eta\{\alpha\mapsto[\![\sigma]\!]_\eta\}\{\beta\mapsto t\}}) \\
= & \quad \langle\text{Lemma 4.4 and } \beta \notin \text{FTV}(\sigma)\rangle \\
 & \text{fix}(\lambda t.[\![\tau']\!]_{\eta\{\beta\mapsto t\}\{\alpha\mapsto[\![\sigma]\!]_{\eta\{\beta\mapsto t\}}\}}) \\
= & \quad \langle\text{Induction Hypothesis}\rangle \\
 & \text{fix}(\lambda t.[\![\tau'\{\alpha := \sigma\}]\!]_{\eta\{\beta\mapsto t\}}) \\
= & \quad \langle\text{Definition}\rangle \\
 & [\![\mathsf{rec}\ \beta.(\tau'\{\alpha := \sigma\})]\!]_\eta \\
= & \\
 & [\![\tau\{\alpha := \sigma\}]\!]_\eta
\end{array}
$$

□

4.3 Relating Interpretation to Subterms

Tying together results about equality rules on terms with the algorithm, which is in terms of automata, requires a correspondance between subtrees of a term's interpretation and the interpretation of its subterms. The purpose of this section is to define and prove this correspondance.

The results in this section are somewhat sensivitive to which alpha variant of a term is considered. The use of these results in the rest of the paper is not so sensivitive to which alpha variant, so without loss of generality, I assume that the alpha variant that is chosen has all binders distinct.

The actual nodes in a term's tree are determined by the parts of the term not involved with recursive quantifiers, the latter only determine the repetitive structure of the tree. Therefore it is important to identify the subterms that determine nodes.

Definition 4.4 *The set* $\text{PROPER}(\tau)$ *is the subterms of* τ *that are not recursive quantifiers nor variables bound by recursive quantifiers of* τ*. Every subterm of* τ *can be mapped to* $\text{PROPER}(\tau)$ *as follows: if* $\sigma \in \text{PROPER}(\tau)$ *then* $\text{PROPER}(\tau, \sigma) = \sigma$*;* $\text{PROPER}(\tau, \mathsf{rec}\ \alpha.\sigma) = \text{PROPER}(\tau, \sigma)$*; if* α *is a subterm of* τ *bound by the subterm* $\mathsf{rec}\ \alpha.\sigma$ *then* $\text{PROPER}(\tau, \alpha) = \text{PROPER}(\tau, \mathsf{rec}\ \alpha.\sigma)$*.*

Lemma 4.11 *If* σ *is a subterm of* τ *then* $\text{PROPER}(\tau, \sigma)$ *is uniquely defined.*

Proof: Let σ_0 be a maximal subterm of τ such that $\text{PROPER}(\tau, \sigma_0)$ does not have a unique definition. Then σ_0 has the form $\mathsf{rec}\ \alpha_1.\sigma_1$ because if it had the form α bound by $\mathsf{rec}\ \alpha.\sigma'$ subterm of τ then $\text{PROPER}(\tau, \sigma_0) = \text{PROPER}(\tau, \mathsf{rec}\ \alpha.\sigma')$ and $\mathsf{rec}\ \alpha.\sigma'$ is larger than σ_0 so $\text{PROPER}(\tau, \mathsf{rec}\ \alpha.\sigma')$ is uniquely defined. Furthermore, σ_1 must either have the form α_1 or the form $\mathsf{rec}\ \alpha_2.\sigma_2$ for otherwise $\text{PROPER}(\tau, \sigma_1)$ would have a unique definition. By induction, there are n and j such that $1 \le j \le n+1$, $\sigma_i = \mathsf{rec}\ \alpha_{i+1}.\sigma_{i+1}$ for $0 \le i \le n$, and $\sigma_{n+1} = \alpha_j$. But then not $\sigma_j \downarrow \alpha_j$, so σ_{j-1} is not a term. $\square$

Next, I associate subterms with the paths of a term's tree.

Definition 4.5 *The path of σ a subterm of τ is:*

$$\begin{array}{lcll} \text{PATH}(\tau,\tau) & = & \epsilon & \\ \text{PATH}(nl(\ell = \vec{\alpha_\ell}.\tau_\ell),\sigma) & = & \ell\text{PATH}(\tau_\ell,\sigma) & \textit{when } \sigma \textit{ a subterm of } \tau_\ell \\ \text{PATH}(\mathsf{rec}\ \alpha.\tau,\sigma) & = & \text{PATH}(\tau,\sigma) & \textit{when } \sigma \textit{ a subterm of } \tau \end{array}$$

The paths of σ in τ is the least set satisfying $\epsilon \in \text{PATHS}(\tau,\tau)$, if $p \in \text{PATHS}(\tau,\alpha)$ and α is bound by $\mathsf{rec}\ \alpha.\sigma$ a subterm of τ then $p \in \text{PATHS}(\tau, \mathsf{rec}\ \alpha.\sigma)$, and if $p \in \text{PATHS}(\tau,\sigma)$ and $p' \in \text{PATH}(\sigma,\sigma')$ then $pp' \in \text{PATHS}(\tau,\sigma')$. Define $\text{PATHS}(\tau) = \cup\{\text{PATHS}(\tau,\sigma) \mid \sigma$ a subterm of $\tau\}$.

Many of the other results can be proven by induction on a certain order. The next lemma defines this order and proves the induction principle.

Lemma 4.12 *For τ a term, let $\sqsubset_\tau$ be the following ordering on pairs of τ's subterms and $\text{PATHS}(\tau)$: $(\mathsf{rec}\ \alpha.\sigma, p) \sqsubset_\tau (\sigma, p)$, $(nl(\ell = \vec{\alpha_\ell}.\sigma_\ell), p) \sqsubset_\tau (\sigma_\ell, p\ell)$, and $(\alpha, p) \sqsubset_\tau (\mathsf{rec}\ \alpha.\sigma, p)$ for α a subterm of σ bound by $\mathsf{rec}\ \alpha.\sigma$. Then $\sqsubset_\tau^*$ is a well founded order and $p \in \text{PATHS}(\tau,\sigma)$ if and only if $(\tau,\epsilon) \sqsubset_\tau^* (\sigma,p)$.*

Proof: Let $(\sigma_1, p_1) \sqsupset_\tau \cdots$ be an infinite descending sequence. Then by definition, $p_1 \ge p_2 \ge \cdots$. Since ordering of paths is well founded there is an i such that $p_i = p_{i+1} = \cdots$. Then by inspection of the definition of the ordering there must be a j and n such that $\sigma_{j+k} = \mathsf{rec}\ \alpha_{j+k}.\sigma_{j+k+1}$ for $0 \le k < n$ and $\sigma_{j+n} = \alpha_j$. But then not $\sigma_{j+1} \downarrow \alpha_j$, and so σ_j is not a well formed term and so neither is τ. If $(\tau,\epsilon) \sqsubset^* \tau(\sigma,p)$ then by induction and the closure properties of $\text{PATHS}(\cdot,\cdot)$, it is easy to show that $p \in \text{PATHS}(\tau,\sigma)$. It is easy to show that $\{(\sigma,p) \mid (\tau,\epsilon) \sqsubset^* \tau(\sigma,p)\}$ is closed under the properties of $\text{PATHS}(\cdot,\cdot)$ and thus that it is a superset of $\{(\sigma,p) \mid p \in \text{PATHS}(\tau,\sigma)\}$. $\square$

Several properties of the path association are needed for latter proofs.

Lemma 4.13

- *If $(\sigma_1,p_1) \sqsubset_\tau (\sigma_2,p_2)$ then either σ_2 is an immediate subterm of σ_1 or σ_2 is an ancestor of σ_1.*
- *If σ is a subterm of σ' is a subterm of τ and $(\tau,\epsilon) \sqsubset_\tau^* (\sigma,p)$ then there exists p' such that $(\tau,\epsilon) \sqsubset_\tau^* (\sigma',p') \sqsubset_\tau^* (\sigma,p)$.*
- *If $(\sigma,p) \sqsubset_\tau (\sigma_1,p_1) \sqsubset_\tau \cdots \sqsubset_\tau (\sigma_n,p_n)$, σ is not one of the σ_i, and σ_n is a subterm of σ then each σ_i is a subterm of σ and σ_1 is the immediate subterm of σ that contains σ_n.*
- *If σ is an occurance of a variable bound along edge ℓ of σ' subterm of τ and $p \in \text{PATHS}(\tau,\sigma)$ then $p = p_1\ell p_2$, $p_1 \in \text{PATHS}(\tau,\sigma')$, and $p' \notin \text{PATHS}(\tau,\sigma')$ for $p_1\ell \le p' \le p$.*

Proof: The first result is immediate from the definition. The second result follows from the first by an easy induction on the ordering in Lemma 4.12. For the third result, by induction and the first result, each σ_i is a subterm of the immediate subterm of σ that contains σ_n. For the fourth result, by Lemma 4.12, $(\tau, \epsilon) \sqsubset_\tau^* (\sigma, p)$. By the second result, there is a p_1 such that $(\tau, \epsilon) \sqsubset_\tau^* (\sigma', p_1) \sqsubset_\tau^* (\sigma, p)$; let p_1 be a maximal such p_1. By induction and the definition of $\sqsubset$, $p_1 \leq p$. Since $\sigma' \neq \sigma$, $(\sigma', p_1) \sqsubset_\tau (\sigma'', p_1') \sqsubset_\tau^* (\sigma, p)$ and again $p_1' \leq p$. By the third result σ' is the immediate subterm of σ' that contains σ. By definition of $\sqsubset$, $p_1' = p_1\ell$. Thus $p = p_1\ell p_2$ for some p_2. That $p' \notin \text{PATHS}(\tau, \sigma')$ for $p_1\ell \leq p' \leq p$ follows by Lemma 4.12 and the fact that p_1 is maximal. □

The first main result is that a subtree of term's tree along path p is the interpretation of a subterm of path p in a certain environment. The next definition defines this environment.

Definition 4.6 *The environment for subterm σ of τ at path p from initial environment η is defined by induction on the ordering in Lemma 4.12 as follows:*

$$\begin{array}{lcl}
\text{ENV}_{\eta,\tau}(\tau, \epsilon) & = & \eta \\
\text{ENV}_{\eta,\tau}(\sigma, p) & = & \text{ENV}_{\eta,\tau}(\sigma', p)\{\alpha \mapsto \text{SUBTREE}([\![\tau]\!]_\eta, p)\} \\
 & & \quad \textit{where } \sigma' = \mathsf{rec}\ \alpha.\sigma \wedge (\sigma', p) \sqsubset_\tau (\sigma, p) \\
\text{ENV}_{\eta,\tau}(\sigma_\ell, p\ell) & = & \text{SHIFT}(\text{ENV}_{\eta,\tau}(\sigma', p), \vec{\alpha_\ell}) \\
 & & \quad \textit{where } \sigma' = nl(\ell = \vec{\alpha_\ell}.\sigma_\ell) \wedge (\sigma', p) \sqsubset_\tau (\sigma_\ell, p\ell) \\
\text{ENV}_{\eta,\tau}(\mathsf{rec}\ \alpha.\sigma, p) & = & \text{ENV}_{\eta,\tau}(\alpha, p) \\
 & & \quad \textit{where } (\alpha, p) \sqsubset_\tau (\mathsf{rec}\ \alpha.\sigma, p)
\end{array}$$

Some properties of this environment are needed to prove the other results.

Lemma 4.14 *If*

- $(\sigma_1, p_1) \sqsubset_\tau \cdots \sqsubset_\tau (\sigma_n, p_n)$,
- $I = \{i \mid 1 \leq i < n \wedge \sigma_i = nl_i(\ell = \vec{\alpha_{i,\ell}}.\sigma_{i,\ell})\}$,
- $p_{i+1} = p_i\ell_i$ *for* $i \in I$,
- $\vec{\alpha} = \vec{\alpha_{i_m,\ell_{i_m}}} \cdots \vec{\alpha_{i_1,\ell_{i_1}}}$ *where* $I = \{i_1, \ldots, i_m\}$ *and* $i_1 < \cdots < i_m$, *and*
- $J = \{j \mid 1 \leq j < n \wedge \sigma_i = \mathsf{rec}\ \beta_j.\sigma_i'\}$

then for some p_j' and n_j:

$$\text{ENV}_{\eta,\tau}(\sigma_2, p_2) = \text{SHIFT}(\text{ENV}_{\eta,\tau}(\sigma_1, p_1), \vec{\alpha})\{\beta_j \mapsto \text{SHIFT}(\text{SUBTREE}([\![\tau]\!]_\eta, p_j'), n_j)\}_{j \in J}$$

Proof: The proof is by induction on the ordering of Lemma 4.12, Lemma 4.1, and definition. □

Here is the first main result.

Lemma 4.15 *If* $p \in \text{PATHS}(\tau, \sigma)$ *then* $\text{SUBTREE}([\![\tau]\!]_\eta, p) = [\![\sigma]\!]_{\text{ENV}_{\eta,\tau}(\sigma,p)}$.

Proof: The proof is by induction on the ordering in Lemma 4.12. In the base case, $p = \epsilon$ and $\sigma = \tau$ so $\text{SUBTREE}([\![\tau]\!]_\eta, p) = [\![\tau]\!]_\eta$ and $\text{ENV}_{\eta,\tau}(\sigma, p) = \eta$ so $[\![\sigma]\!]_{\text{ENV}_{\eta,\tau}(\sigma,p)} = [\![\tau]\!]_\eta$, as required. Consider the cases for the inductive step:

$(\mathsf{rec}\ \alpha.\sigma, p) \sqsubset_\tau (\sigma, p)$**:**

$$\begin{array}{cl} & \text{SUBTREE}([\![\tau]\!]_\eta, p) \\ = & \quad \langle\text{Induction Hypothesis}\rangle \\ & [\![\mathsf{rec}\ \alpha.\sigma]\!]_{\text{ENV}_{\eta,\tau}(\mathsf{rec}\ \alpha.\sigma,p)} \\ = & \quad \langle\text{Definition}\rangle \\ & [\![\sigma]\!]_{\text{ENV}_{\eta,\tau}(\mathsf{rec}\ \alpha.\sigma,p)\{\alpha \mapsto [\![\mathsf{rec}\ \alpha.\sigma]\!]_{\text{ENV}_{\eta,\tau}(\mathsf{rec}\ \alpha.\sigma,p)}\}} \\ = & \quad \langle\text{Induction Hypothesis}\rangle \\ & [\![\sigma]\!]_{\text{ENV}_{\eta,\tau}(\mathsf{rec}\ \alpha.\sigma,p)\{\alpha \mapsto \text{SUBTREE}([\![\tau]\!]_\eta,p)\}} \\ = & \quad \langle\text{Definition}\rangle \\ & [\![\sigma]\!]_{\text{ENV}_{\eta,\tau}(\sigma,p)} \end{array}$$

$(nl(\ell = \vec{\alpha_\ell}.\sigma_\ell), p) \sqsubset_\tau (\sigma_\ell, p\ell)$**:**

$$\begin{array}{cl} & \text{SUBTREE}([\![\tau]\!]_\eta, p) \\ = & \quad \langle\text{Induction Hypothesis}\rangle \\ & [\![nl(\ell = \vec{\alpha_\ell}.\sigma_\ell, p)]\!]_{\text{ENV}_{\eta,\tau}(nl(\ell=\vec{\alpha_\ell}.\sigma_\ell,p),p)} \\ = & \quad \langle\text{Definition}\rangle \\ & \mathsf{nl}(nl, \ell = [\![\sigma_\ell]\!]_{\text{SHIFT}(\text{ENV}_{\eta,\tau}(nl(\ell=\vec{\alpha_\ell}.\sigma_\ell,p),p),\vec{\alpha_\ell})}) \\ = & \quad \langle\text{Definition}\rangle \\ & \mathsf{nl}(nl, \ell = [\![\sigma_\ell]\!]_{\text{ENV}_{\eta,\tau}(\sigma_\ell,p\ell)}) \end{array}$$

By Lemma 3.6, $\text{SUBTREE}([\![\tau]\!]_\eta, p\ell) = [\![\sigma_\ell]\!]_{\text{ENV}_{\eta,\tau}(\sigma_\ell,p\ell)}$, as required.

$(\alpha, p) \sqsubset_\tau (\mathsf{rec}\ \alpha.\sigma, p)$**:** By Lemma 4.13, there is a p' such that $(\tau, \epsilon) \sqsubset_\tau^* (\mathsf{rec}\ \alpha.\sigma, p') \sqsubset_\tau^* (\alpha, p)$. Let p' be a maximal such path. By Lemmas 4.14 and 4.13,

$$\begin{array}{rcl} \text{ENV}_{\eta,\tau}(\alpha, p) & = & \text{SHIFT}(\text{ENV}_{\eta,\tau}(\mathsf{rec}\ \alpha.\sigma, p')\{\alpha := \text{SUBTREE}([\![\tau]\!]_\eta, p')\}, \vec{\beta}) \\ & & \{\beta_i \mapsto \text{SHIFT}(\text{SUBTREE}([\![\tau]\!]_\eta, p_i), n_i)\} \end{array}$$

for some vector $\vec{\beta}$ of binders in σ and some variables β_i bound by recursive quantifiers in σ. Since α is free in σ it is different from each $\vec{\beta}$ and each β_i so $\text{ENV}_{\eta,\tau}(\alpha, p)(\alpha) = \text{SHIFT}(\text{SUBTREE}([\![\tau]\!]_\eta, p'), |\vec{\beta}|)$ (1). Since each $\vec{\beta}$ and each β_i are binders in σ they are not free in σ, thus $\text{SHIFT}(\text{ENV}_{\eta,\tau}(\mathsf{rec}\ \ \alpha.\sigma, p'), \vec{\beta})(\gamma) = \text{ENV}_{\eta,\tau}(\mathsf{rec}\ \ \alpha.\sigma, p)(\gamma)$ for $\gamma \in$

$\textsc{ftv}(\mathsf{rec}\ \alpha.\sigma)$ (2).

$$\begin{array}{ll} & \textsc{subtree}([\![\tau]\!]_\eta, p) \\ = & \quad \langle \text{Induction Hypothesis} \rangle \\ & [\![\alpha]\!]_{\textsc{env}_{\eta,\tau}(\alpha,p)} \\ = & \quad \langle \text{Definition and (1)} \rangle \\ & \textsc{shift}(\textsc{subtree}([\![\tau]\!]_\eta, p'), |\vec{\beta}|) \\ = & \quad \langle \text{Induction Hypothesis} \rangle \\ & \textsc{shift}([\![\mathsf{rec}\ \alpha.\sigma]\!]_{\textsc{env}_{\eta,\tau}(\mathsf{rec}\ \alpha.\sigma,p')}, |\vec{\beta}|) \\ = & \quad \langle \text{Lemma 4.9} \rangle \\ & [\![\mathsf{rec}\ \alpha.\sigma]\!]_{\textsc{shift}(\textsc{env}_{\eta,\tau}(\mathsf{rec}\ \alpha.\sigma,p'),\vec{\beta})} \\ = & \quad \langle \text{Lemma 4.4 and (2)} \rangle \\ & [\![\mathsf{rec}\ \alpha.\sigma]\!]_{\textsc{env}_{\eta,\tau}(\mathsf{rec}\ \alpha.\sigma,p)} \end{array}$$

□

The next main result, is a correspondance between the binding along a path in a term's tree and the shifting of the environment.

Lemma 4.16 *If* $\textsc{env}_{\eta,\tau}(\sigma, p) = \textsc{shift}(\textsc{env}_{\eta,\tau}(\tau, \epsilon), \vec{\alpha})\{\beta_i \mapsto t_i\}$ *then* $\textsc{bind}([\![\tau]\!]_\eta, p) = |\vec{\alpha}|$.

Proof: The proof follows by a straightfoward induction from Lemmas 4.14 and 4.15. □

Here is the third main result, which is the correspondance necessary to prove the results that tie the report together. It states that the node (subtree in the case of a free variable) at a path in a term's tree is determined in a certain way by the proper subterm for that path.

Lemma 4.17 *If* $p \in \textsc{paths}(\tau, \sigma)$ *then:*

- *If* σ *has the form* $nl(\cdots)$ *then* $[\![\tau]\!]_\eta(p) = \mathrm{in}_1(nl)$.
- *If* σ *is a variable free in* τ *then* $\textsc{subtree}([\![\tau]\!]_\eta, p) = \textsc{shift}(\eta(\sigma), \textsc{bind}([\![\tau]\!]_\eta, p))$.
- *If* σ *is bound by the nth binder of edge* ℓ *of* σ' *subterm of* τ, $p = p_1 \ell p_2$, $p_1 \in \textsc{paths}(\tau, \sigma')$, *and* $p' \notin \textsc{paths}(\tau, \sigma')$ *for* $p_1\ell \le p' \le p$ *then* $[\![\tau]\!]_\eta(p) = \mathrm{in}_2(\textsc{bind}([\![\tau]\!]_\eta, p_1\ell \to p) + n)$.

Proof: Let $p \in \textsc{paths}(\tau, \sigma)$. By Lemma 4.15, $\textsc{subtree}([\![\tau]\!]_\eta, p) = [\![\sigma]\!]_{\textsc{env}_{\eta,\tau}(\sigma,p)}$. Consider the cases. If $\sigma = nl(\ell = \vec{\alpha_\ell}.\sigma_\ell)$ then $[\![\sigma]\!]_{\textsc{env}_{\eta,\tau}(\sigma,p)} = \mathsf{nl}(nl, \ell = [\![\sigma_\ell]\!]_{\textsc{shift}(\textsc{env}_{\eta,\tau}(\sigma,p),\vec{\alpha_\ell})})$. Thus $[\![\tau]\!]_\eta(p) = \textsc{subtree}([\![\tau]\!]_\eta, p)(\epsilon) = \mathrm{in}_1(nl)$, as required. If σ is an occurance of a free variable of τ then by Lemma 4.14, $\textsc{env}_{\eta,\tau}(\sigma, p) = \textsc{shift}(\eta, \vec{\alpha})\{\beta_i \mapsto t_i\}$ for $\vec{\alpha}$ binders in τ and β_i bound by recursive quantifiers of τ. Since σ is free in τ, $\textsc{shift}(\eta, \vec{\alpha})\{\beta_i \mapsto t_i\}(\sigma) = \textsc{shift}(\eta(\sigma), |\vec{\alpha}|)$. By Lemma 4.16, $|\vec{\alpha}| = \textsc{bind}([\![\tau]\!]_\eta, p)$, as required. If σ is an occurance of a variable bound by the nth binder of edge ℓ of σ' subterm of τ then by Lemma 4.13, $p = p_1\ell p_2$, $p_1 \in \textsc{paths}(\tau, \sigma')$, and $p' \notin \textsc{paths}(\tau, \sigma')$ for $p_1\ell \le p' \le p$. By Lemma 4.14, $\textsc{env}_{\eta,\tau}(\sigma', p_1\ell) = \textsc{shift}(\eta, \vec{\alpha_1}\vec{\alpha_2})\{\beta_i \mapsto t_i\}$, $\textsc{env}_{\eta,\tau}(\sigma, p) = \textsc{shift}(\eta, \vec{\alpha_3}\vec{\alpha_1}\vec{\alpha_2})\{\gamma_i \mapsto t'_i\}$, and $\sigma = \alpha_{1n}$. Also by Lemma 4.13, $\vec{\alpha_3}$ consists of binders of subterms of σ'. By assumption these binders are distinct from the binders of σ'

itself, so are different from $\vec{\alpha_1}$. Thus, $[\![\sigma]\!]_{\mathrm{ENV}_{\eta,\tau}(\sigma,p)} = \textsc{shift}(\mathsf{var}(n), |\vec{\alpha_3}|)$. By Lemma 4.16, $|\vec{\alpha_3}| = \textsc{bind}([\![\tau]\!]_\eta, p_1\ell \to p)$. Thus $[\![\tau]\!]_\eta(p) = \textsc{subtree}([\![\tau]\!]_\eta, p)(\epsilon) = \textsc{shift}(\mathsf{var}(n), |\vec{\alpha_3}|)(\epsilon) = \mathrm{in}_2(n + \textsc{bind}([\![\tau]\!]_\eta, p_1\ell \to p))$, as required. $\square$

4.4 Terms for Trees

The completeness proof is based on canonical terms for regular trees. The purpose of this section is to define these terms and prove some properties about them.

Canonical terms will be defined with respect to certain kinds of environments—those that map term variables to the free variables of the tree in question. Completeness will be proven with respect to certain kinds of environments—those that map different terms to different trees.

Definition 4.7 *If $N \subseteq \mathbb{N}$ then $\textsc{covers}(\eta, N)$ exactly when $\forall n \in N : \exists \alpha : \eta(\alpha) = \mathsf{var}(n)$. Environment η is distinguishing exactly when $\eta(\alpha) = \mathsf{var}(f(\alpha))$ for some injective function f from TVar to $\mathbb{N}$.*

Definition 4.8 *If $t \in RegTree$, η covers $\textsc{ftv}(t)$, and R is a nonoverlapping equality of t's subtrees with a finite number of equivalence classes then:*

- *If f maps $[R]$ to TVar, f maps $[p]_R, \ell$ to a sequence of TVar of length $\textsc{bind}(nl, \ell)$ when $t(p) = \mathrm{in}_1(nl)$ and $\ell \in \textsc{labels}(nl)$, S is a subset of $[R]$, and $p \in \mathrm{dom}(t)$ then:*

$$\mathsf{T}^{t,\eta,R,f}_{S,p} = \begin{cases} f([p]_R) & [p]_R \in S \\ \mathsf{rec}\ f([p]_R).\mathsf{S}^{t,\eta,R,f}_{S\cup\{[p]_R\},p} & [p]_R \notin S \end{cases}$$

$$\mathsf{S}^{t,\eta,R,f}_{S,p} = \begin{cases} \beta & \textsc{varof}(t,p) = \mathsf{free}(n) \wedge \eta(\beta) = \mathsf{var}(n) \\ f([p']_R, \ell)_n & \textsc{varof}(t,p) = \mathsf{bound}(p', \ell, n) \\ nl(\ell = f([p]_R, \ell).\mathsf{T}^{t,\eta,R,f}_{S,p\ell}) & t(p) = \mathrm{in}_1(nl) \end{cases}$$

- $\textsc{termof}_\eta(t, R) = \mathsf{T}^{t,\eta,R,f}_{\emptyset,\epsilon}$ *for some appropriate f that maps to fresh variables*
- $\textsc{termof}_\eta(t) = \textsc{termof}_\eta(t, \textsc{eqst}(t))$

The term of a tree is defined so it maps to the tree.

Lemma 4.18 *If $t \in RegTree$, $\textsc{covers}(\eta, \textsc{ftv}(t))$, and R is a nonoverlapping equivalence of t's subtrees with finite number of equivalence classes then $[\![\textsc{termof}_\eta(t, R)]\!]_\eta = t$.*

Proof: Let $t \in RegTree$, $\textsc{covers}(\eta, \textsc{ftv}(t))$, and R be a nonoverlapping equivalence of t's subtrees with finite number of equivalence classes. Let f be such that $\textsc{termof}_\eta(t, R) = \mathsf{T}^{t,\eta,R,f}_{\emptyset,\epsilon}$. Let v_ϵ be the empty sequence of variables, and $v_{p\ell} = f([p]_R, \ell)v_p$. A simple induction shows that $\text{BIND}(t,p) = |v_p|$. Let $\min(R,p)$ be the least element of $\{p' \mid p' \leq p \wedge p' \in [p]_R\}$. Let $S_p = \{[p']_R \mid p' < p\}$ and $S'_p = \{[p']_R \mid p' \leq p\}$. Let:

$$\begin{array}{rcl}\eta_p & = & \text{SHIFT}(\eta, v_p)\{f([p']_R) \mapsto \text{SHIFT}(\text{SUBTREE}(t, \min(R,p')), \text{BIND}(t, \min(R,p') \to p))\}_{p'<p} \\ \eta'_p & = & \text{SHIFT}(\eta, v_p)\{f([p']_R) \mapsto \text{SHIFT}(\text{SUBTREE}(t, \min(R,p')), \text{BIND}(t, \min(R,p') \to p))\}_{p'\leq p}\end{array}$$

With these definitions, the desired result is that $[\![\mathsf{T}^{t,\eta,R,f}_{S_\epsilon,\epsilon}]\!]_{\eta_\epsilon} = \text{SUBTREE}(t,\epsilon)$. Therefore the rest of the proof shows that $[\![\mathsf{T}^{t,\eta,R,f}_{S_p,p}]\!]_{\eta_p} = [\![\mathsf{S}^{t,\eta,R,f}_{S'_p,p}]\!]_{\eta'_p} = \text{SUBTREE}(t,p)$ for p such that $\neg\exists p_1, p_2 : p_1 < p_2 < p \wedge p_1 \; R \; p_2$ for T and $\neg\exists p_1, p_2 : p_1 < p_2 \leq p \wedge p_1 \; R \; p_2$ for S. The proof proceeds by induction on the size of the terms. Consider the cases.

T, $[p]_R \in S_p$: In this case $\mathsf{T}^{t,\eta,R,f}_{S_p,p} = f([p]_R)$, so:

$$[\![\mathsf{T}^{t,\eta,R,f}_{S_p,p}]\!]_{\eta_p} = \eta_p(f([p]_R)) = \text{SHIFT}(\text{SUBTREE}(t, \min(R,p)), \text{BIND}(t, \min(R,p) \to p))$$

Note that $\min(R,p) \; R \; p$, $\min(R,p) < p$, and $\neg\exists p_1, p_2 : p_1 < p_2 < p \wedge p_1 \; R \; p_2$. By Lemma 3.9, $\text{SHIFT}(\text{SUBTREE}(t, \min(R,p)), \text{BIND}(t, \min(R,p) \to p)) = \text{SUBTREE}(t,p)$.

T, $[p]_R \notin S_p$: In this case, $\mathsf{T}^{t,\eta,R,f}_{S_p,p} = \mathsf{rec}\; f([p]_R).\mathsf{S}^{t,\eta,R,f}_{S'_p,p}$. Since $[p]_R \notin S_p$, there is no $p' < p$ such that $p' \; R \; p$, that is, $\min(R,p) = p$ and $\neg\exists p_1, p_2 : p_1 < p_2 \leq p \wedge p_1 \; R \; p_2$. By the induction hypothesis, $\text{SUBTREE}(t,p) = [\![\mathsf{S}^{t,\eta,R,f}_{S'_p,p}]\!]_{\eta'_p}$. Clearly, $\eta'_p = \eta_p\{f([p]_R) \mapsto \text{SUBTREE}(t,p)\}$. So by definition, $\text{SUBTREE}(t,p) = [\![f([p]_R).\mathsf{S}^{t,\eta,R,f}_{S'_p,p}]\!]_{\eta_p}(\text{SUBTREE}(t,p))$. By uniqueness of fixed points, $\text{SUBTREE}(t,p) = \text{fix}([\![f([p]_R).\mathsf{S}^{t,\eta,R,f}_{S'_p,p}]\!]_{\eta_p}) = [\![\mathsf{T}^{t,\eta,R,f}_{S_p,p}]\!]_{\eta_p}$, as required.

S, $\text{VAROF}(t,p) = \mathsf{free}(n)$: In this case, $\mathsf{S}^{t,\eta,R,f}_{S'_p,p} = \beta$ where $\eta(\beta) = \mathsf{var}(n)$. So $[\![\mathsf{S}^{t,\eta,R,f}_{S'_p,p}]\!]_{\eta'_p} = \eta'_p(\beta) = \text{SHIFT}(\eta(\beta), |v_p|) = \text{SHIFT}(\eta(\beta), \text{BIND}(t,p)) = \mathsf{var}(n + \text{BIND}(t,p))$. The result follows by Lemma 3.6.

S, $\text{VAROF}(t,p) = \mathsf{bound}(p', \ell, n)$: In this case, $\mathsf{S}^{t,\eta,R,f}_{S'_p,p} = f([p']_R, \ell)_n$. By definition, $[\![\mathsf{S}^{t,\eta,R,f}_{S'_p,p}]\!]_{\eta'_p} = \eta'_p(f([p']_R, \ell)_n)$. Since R is nonoverlapping, p' is not related to any p'' such $p' < p'' < p$. Thus $v_p = \vec{\alpha} v_{p'\ell}$ for some $\vec{\alpha}$ that does not contain $f([p']_R, \ell)_n$. Since $|v_{p'\ell}| = \text{BIND}(t, p'\ell)$ and $|v_p| = \text{BIND}(t,p)$, $|\vec{\alpha}| = \text{BIND}(t, p'\ell \to p)$. Thus $\eta'_p(f([p']_R, \ell)_n) = \mathsf{var}(n + \text{BIND}(t, p'\ell \to p))$. The result follows by Lemma 3.6.

S, $t(p) = \text{in}_1(nl)$: In this case, $\mathsf{S}^{t,\eta,R,f}_{S'_p,p} = nl(\ell = f([p]_R, \ell).\mathsf{T}^{t,\eta,R,f}_{S_{p\ell},p\ell})$, so $[\![\mathsf{S}^{t,\eta,R,f}_{S'_p,p}]\!]_{\eta'_p} = \mathsf{nl}(nl, \ell = [\![\mathsf{T}^{t,\eta,R,f}_{S_{p\ell},p\ell}]\!]_{\text{SHIFT}(\eta'_p, f([p]_R, \ell))})$. Clearly $\text{SHIFT}(\eta'_p, f([p]_R, \ell)) = \eta_{p\ell}$, so $[\![\mathsf{S}^{t,\eta,R,f}_{S'_p,p}]\!]_{\eta'_p} = \mathsf{nl}(nl, \ell = [\![\mathsf{T}^{t,\eta,R,f}_{S_{p\ell},p\ell}]\!]_{\eta_{p\ell}})$. Since $\neg\exists p_1, p_2 : p_1 < p_2 \leq p \wedge p_1 \; R \; p_2$, then $\neg\exists p_1, p_2 : p_1 < p_2 < p\ell \wedge p_1 \; R \; p_2$. By the induction hypothesis, $[\![\mathsf{T}^{t,\eta,R,f}_{S_{p\ell},p\ell}]\!]_{\eta_{p\ell}} = \text{SUBTREE}(t, p\ell)$. Thus, $[\![\mathsf{S}^{t,\eta,R,f}_{S'_p,p}]\!]_{\eta'_p} = \mathsf{nl}(nl, \ell = \text{SUBTREE}(t, p\ell))$ and the result follows by Lemma 3.6.

□

Theorem 4.19 *If $t \in \mathit{RegTree}$ and $\text{COVERS}(\eta, \text{FTV}(t))$ then $[\![\text{TERMOF}_\eta(t)]\!]_\eta = t$.*

4.5 Term-Equality Rules

The term-equality rules derive judgements of the form $\vdash \tau_1 = \tau_2$ intended to mean that terms τ_1 and τ_2 are equal. The rules are essentially those of Amadio and Cardelli:

$$\text{(eqsym)}\ \frac{\vdash \tau_2 = \tau_1}{\vdash \tau_1 = \tau_2} \qquad \text{(eqtrans)}\ \frac{\vdash \tau_1 = \tau_2 \quad \vdash \tau_2 = \tau_3}{\vdash \tau_1 = \tau_3} \qquad \text{(eqvar)}\ \frac{}{\vdash \alpha = \alpha}$$

$$\text{(eqnl)}\ \frac{\vdash \tau_\ell = \sigma_\ell}{\vdash nl(\ell = \vec{\alpha_\ell}.\tau_\ell) = nl(\ell = \vec{\alpha_\ell}.\sigma_\ell)} \qquad \text{(eqrec)}\ \frac{\vdash \tau = \sigma}{\vdash \mathsf{rec}\ \alpha.\tau = \mathsf{rec}\ \alpha.\sigma}$$

$$\text{(eqroll)}\ \frac{}{\vdash \mathsf{rec}\ \alpha.\tau = \tau\{\alpha := \mathsf{rec}\ \alpha.\tau\}} \quad \text{(equnq)}\ \frac{\vdash \tau_1 = \sigma\{\alpha := \tau_1\} \quad \vdash \tau_2 = \sigma\{\alpha := \tau_2\}}{\vdash \tau_1 = \tau_2}\ (\sigma \downarrow \alpha)$$

The first two rules express that equality is symmetric and transitive. The next three rules express that equality is closed under all the term constructors and that equality is reflexive. Together these five rules make equality a congruence relation. The last two rules are the interesting ones. Rule (eqroll) expresses that $\mathsf{rec}\ \alpha.\tau$ is the fixed point of the mapping of α to τ, and is the rule from the introduction that defines the equirecursive approach. Rule (equnq) expresses the fact that contractive mappings have unique fixed points. The hypotheses expresses that τ_1 and τ_2 are a fixed points of the mapping α to σ; the side condition expresses that the mapping is contractive; and the conclusion expresses that the two fixed points are equal.

4.5.1 Soundness

It is straightforward to prove the rules sound, that is, that provably equal terms map to the same tree.

Theorem 4.20 (Soundness) *If* $\vdash \tau_1 = \tau_2$ *then* $[\![\tau_1]\!]_\eta = [\![\tau_2]\!]_\eta$ *for all* $\eta \in \mathit{Env}$.

Proof: The proof is by induction on the derivation of $\vdash \tau_1 = \tau_2$. Let $\eta \in \mathit{Env}$. Consider the last rule used in the proof.

(eqsym): By the induction hypothesis $[\![\tau_2]\!]_\eta = [\![\tau_1]\!]_\eta$, as required.

(eqtrans): By the induction hypothesis $[\![\tau_1]\!]_\eta = [\![\tau_2]\!]_\eta$ and $[\![\tau_2]\!]_\eta = [\![\tau_3]\!]_\eta$. Thus $[\![\tau_1]\!]_\eta = [\![\tau_3]\!]_\eta$, as required.

(eqvar): By definition, $[\![\tau_1]\!]_\eta = \eta(\alpha) = [\![\tau_2]\!]_\eta$, as required.

(eqnl): In this case $\tau_1 = nl(\ell = \vec{\alpha_\ell}.\tau_{1\ell})$, $\tau_2 = nl(\ell = \vec{\alpha_\ell}.\tau_{2\ell})$, and $\vdash \tau_{1\ell} = \tau_{2\ell}$. By the induction hypothesis, $[\![\tau_{1\ell}]\!]_{\mathrm{SHIFT}(\eta,\vec{\alpha_\ell})} = [\![\tau_{2\ell}]\!]_{\mathrm{SHIFT}(\eta,\vec{\alpha_\ell})}$. So $\mathsf{nl}(nl, \ell = [\![\tau_{1\ell}]\!]_{\mathrm{SHIFT}(\eta,\vec{\alpha_\ell})}) = \mathsf{nl}(nl, \ell = [\![\tau_{2\ell}]\!]_{\mathrm{SHIFT}(\eta,\vec{\alpha_\ell})})$. By definition, $[\![\tau_1]\!]_\eta = [\![\tau_2]\!]_\eta$, as required.

(eqrec): In this case $\tau_1 = \mathsf{rec}\ \alpha.\tau_1'$ and $\tau_2 = \mathsf{rec}\ \alpha.\tau_2'$. By the induction hypothesis, $[\![\tau_1']\!]_{\eta\{\alpha\mapsto t\}} = [\![\tau_2']\!]_{\eta\{\alpha\mapsto t\}}$. So $\lambda t.[\![\tau_1']\!]_{\eta\{\alpha\mapsto t\}} = \lambda t.[\![\tau_2']\!]_{\eta\{\alpha\mapsto t\}}$. So $\mathrm{fix}(\lambda t.[\![\tau_1']\!]_{\eta\{\alpha\mapsto t\}}) = \mathrm{fix}(\lambda t.[\![\tau_2']\!]_{\eta\{\alpha\mapsto t\}})$. Thus $[\![\tau_1]\!]_\eta = [\![\tau_2]\!]_\eta$ as required.

(eqroll): In this case $\tau_1 = \mathsf{rec}\ \alpha.\tau$ and $\tau_2 = \tau\{\alpha := \tau_1\}$. Then:

$$
\begin{array}{ll}
& [\![\tau_1]\!]_\eta \\
= & \quad \langle \text{Definition} \rangle \\
& \text{fix}(\lambda t.[\![\tau]\!]_{\eta\{\alpha \mapsto t\}}) \\
= & \quad \langle \text{Fixed point property} \rangle \\
& [\![\tau]\!]_{\eta\{\alpha \mapsto \text{fix}(\lambda t.[\![\tau]\!]_{\eta\{\alpha \mapsto t\}})\}} \\
= & \quad \langle \text{Definition} \rangle \\
& [\![\tau]\!]_{\eta\{\alpha \mapsto [\![\tau_1]\!]_\eta\}} \\
= & \quad \langle \text{Lemma 4.10} \rangle \\
& [\![\tau\{\alpha := \tau_1\}]\!]_\eta \\
= & \\
& [\![\tau_2]\!]_\eta
\end{array}
$$

(equnq): In this case there are σ and α such that $\sigma \downarrow \alpha$, $\vdash \tau_1 = \sigma\{\alpha := \tau_1\}$, and $\vdash \tau_2 = \sigma\{\alpha := \tau_2\}$. By the induction hypothesis, $[\![\tau_1]\!]_\eta = [\![\sigma\{\alpha := \tau_1\}]\!]_\eta$ and $[\![\tau_2]\!]_\eta = [\![\sigma\{\alpha := \tau_2\}]\!]_\eta$. By Lemma 4.10, $[\![\tau_1]\!]_\eta = [\![\alpha.\sigma]\!]_\eta([\![\tau_1]\!]_\eta)$ and $[\![\tau_2]\!]_\eta = [\![\alpha.\sigma]\!]_\eta([\![\tau_2]\!]_\eta)$. Thus $[\![\tau_1]\!]_\eta$ and $[\![\tau_2]\!]_\eta$ are fixed points of $[\![\alpha.\sigma]\!]_\eta$. Since $\sigma \downarrow \alpha$, by Lemma 4.3, $[\![\alpha.\sigma]\!]_\eta$ is contractive, so by Banach's theorem, has a unique fixed point. Thus $[\![\tau_1]\!]_\eta = [\![\tau_2]\!]_\eta$, as required.

□

4.5.2 Completeness

It is more difficult to show the converse, that terms that map to the same tree are provably equal. Amadio and Cardelli showed completeness by defining something called *systems of equations*. Unfortunately, it seems very difficult to define systems of equations for second-order terms. So I use a different approach to showing completeness, which also works in the first-order setting. The idea is to show that a term is provably equal to the canonical term of its tree. Completeness then follows by transitivity.

The proof requires two properties of provable equality—that equality is reflexive, and it is closed under substitution.

Lemma 4.21 *Rule (eqref) is a derivable, where:*

$$
\textit{(eqref)}\ \frac{}{\vdash \tau = \tau}
$$

Proof: The proof is by induction over the structure of τ using rules (eqvar), (eqnl), and (eqrec). □

Lemma 4.22 *If $\vdash \tau_1 = \tau_2$ then $\vdash \tau_1\{\rho\} = \tau_2\{\rho\}$ for ρ any substitution of terms for term variables.*

Proof: The proof is a straightfoward induction on the derivation of $\vdash \tau_1 = \tau_2$. In the case of the rule (eqvar), derived rule (eqref) is used. □

Here is the main technical result, that a term is provably equal to its tree's term.

Lemma 4.23 *If η is distinguishing then* $\vdash \text{TERMOF}_\eta([\![\tau]\!]_\eta) = \tau$.

Proof: Fix a distinguishing η and τ. Let f be such that $\text{TERMOF}_\eta([\![\tau]\!]_\eta) = \mathsf{T}^{[\![\tau]\!]_\eta,\eta,\text{EQST}([\![\tau]\!]_\eta),f}_{\emptyset,\epsilon}$.

The first step of the proof defines for each subterm σ of τ, a path p_σ, substitution ρ_σ, environment η_σ, natural n_σ, and term T_σ by induction on the depth of σ in τ as follows. Let $p_\tau = \epsilon$, ρ_τ be the empty substitution, $\eta_\tau = \eta$, and $n_\tau = 0$. If σ is a subterm of τ of the form $nl(\ell = \vec{\alpha_\ell}.\sigma_\ell)$ and $f([p_\sigma], \ell) = \vec{\alpha_\ell}$ (without loss of generality assume an α-variant of τ that makes this true) then $p_{\sigma_\ell} = p_\sigma\ell$, $\rho_{\sigma_\ell} = \rho_\sigma$, $\eta_{\sigma_\ell} = \text{SHIFT}(\eta_\sigma, \vec{\alpha_\ell})$, and $n_{\sigma_\ell} = n_\sigma + |\vec{\alpha_\ell}|$. If σ is a subterm of τ of the form $\mathsf{rec}\ \alpha.\sigma'$ and α is fresh then $p_{\sigma'} = p_\sigma$, $\rho_{\sigma'} = \rho_\sigma, \alpha := \mathsf{T}_\sigma$, $\eta_{\sigma'} = \eta_\sigma$, and $n_{\sigma'} = n_\sigma$. If σ is a subterm of τ and if $\sigma_1, \ldots, \sigma_n$ are the sequence of proper subterms that lead from τ to σ then define $S_\sigma = \{[p_{\sigma_1}], \ldots, [p_{\sigma_{n-1}}]\}$ and $\mathsf{T}_\sigma = \mathsf{T}^{[\![\tau]\!]_\eta,\eta,\text{EQST}([\![\tau]\!]_\eta),f}_{S_\sigma,p_\sigma}\{f([p_{\sigma_1}]) := \mathsf{T}_{\sigma_1}, \ldots, f([p_{\sigma_{n-1}}]) := \mathsf{T}_{\sigma_n}\}$.

T_σ satisfies two important properties. First, if $\sigma = \mathsf{rec}\ \alpha.\sigma'$ then $\mathsf{T}_\sigma = \mathsf{T}_{\sigma'}$ (1). Second, by a simple induction, if σ_1 is a subterm of σ_2 and $p_{\sigma_2} \in [p_{\sigma_1}]$ then $\mathsf{T}_{\sigma_1} = \mathsf{T}_{\sigma_2}$ (2).

The next step of the proof shows for all σ subterm of τ that $\text{SUBTREE}([\![\tau]\!]_\eta, p_\sigma) = [\![\mathsf{T}_\sigma]\!]_{\eta_\sigma} = [\![\sigma\{\rho_\sigma\}]\!]_{\eta_\sigma}$ and $\text{BIND}([\![\tau]\!]_\eta, p_\sigma) = n_\sigma$. The proof is by induction on the depth of σ in τ. For τ itself, $\text{SUBTREE}([\![\tau]\!]_\eta, p_\tau) = \text{SUBTREE}([\![\tau]\!]_\eta, \epsilon) = [\![\tau]\!]_\eta$, $[\![\mathsf{T}_\tau]\!]_{\eta_\tau} = [\![\mathsf{T}^{[\![\tau]\!]_\eta,\eta,\text{EQST}([\![\tau]\!]_\eta),f}_{\emptyset,\epsilon}]\!]_\eta = [\![\text{TERMOF}_\eta([\![\tau]\!]_\eta)]\!]_\eta = [\![\tau]\!]_\eta$, $[\![\tau\{\rho_\tau\}]\!]_{\eta_\tau} = [\![\tau]\!]_\eta$, $\text{BIND}([\![\tau]\!]_\eta, p_\tau) = \text{BIND}([\![\tau]\!]_\eta, \epsilon) = 0$, and $n_\tau = 0$, as required. Assume the property holds for σ to show that it holds for σ's subterms. Consider the cases for σ:

$\sigma = nl(\ell = \vec{\alpha_\ell}.\sigma_\ell)$**:** In this case $\sigma\{\rho_\sigma\} = nl(\ell = \vec{\alpha_\ell}.\sigma_\ell\{\rho_\sigma\})$ (recall that $\vec{\alpha_\ell}$ is fresh, since it comes from f). Thus $[\![\sigma\{\rho_\sigma\}]\!]_{\eta_\sigma} = \mathsf{nl}(nl, \ell = [\![\sigma_\ell\{\rho_\sigma\}]\!]_{\text{SHIFT}(\eta_\sigma,\vec{\alpha_\ell})})$. By the induction hypothesis and definition, $\text{SUBTREE}([\![\tau]\!]_\eta, p_\sigma) = \mathsf{nl}(nl, \ell = [\![\sigma_\ell\{\rho_{\sigma_\ell}\}]\!]_{\eta_{\sigma_\ell}})$. Thus, $[\![\tau]\!]_\eta(p_\sigma) = \text{in}_1(nl)$ $(*)$ and $\text{SUBTREE}([\![\tau]\!]_\eta, p_\sigma\ell) = [\![\sigma_\ell\{\rho_{\sigma_\ell}\}]\!]_{\eta_{\sigma_\ell}}$. By definition, the latter is equivalent to $\text{SUBTREE}([\![\tau]\!]_\eta, p_{\sigma_\ell}) = [\![\sigma_\ell\{\rho_{\sigma_\ell}\}]\!]_{\eta_{\sigma_\ell}}$, which is part of the desired result. By definition $\text{BIND}([\![\tau]\!]_\eta, p_{\sigma_\ell}) = \text{BIND}([\![\tau]\!]_\eta, p_\sigma\ell) = \text{BIND}([\![\tau]\!]_\eta, p_\sigma) + \text{BIND}(nl, \ell)$. By the induction hypothesis and noting that $\text{BIND}(nl, \ell) = |\vec{\alpha_\ell}|$ by syntactic constraints, this is equal to $n_\sigma + |\vec{\alpha_\ell}| = n_{\sigma_\ell}$, which is another part of the desired result.

Let σ' be the largest proper subterm of τ that contains σ and such that $p_{\sigma'} \in [p_\sigma]$. Then by $(*)$ and the definition of equivalence of subtrees, $[\![\tau]\!]_\eta(p_{\sigma'}) = \text{in}_1(nl)$. Also $[p'_\sigma] \notin S_{\sigma'}$, so by definition, $\mathsf{T}_{\sigma'} = \mathsf{rec}\ f([p_{\sigma'}]).nl(\ell = \vec{\alpha_\ell}.\tau_\ell)$ where $\tau_\ell = \mathsf{T}^{[\![\tau]\!]_\eta,\eta,\text{EQST}([\![\tau]\!]_\eta),f}_{S_{\sigma'}\cup\{[p_{\sigma'}]\},p_{\sigma'}\ell}\{\rho\}$ and $\mathsf{T}_{\sigma'} = \mathsf{T}^{[\![\tau]\!]_\eta,\eta,\text{EQST}([\![\tau]\!]_\eta),f}_{S_{\sigma'},p_{\sigma'}}\{\rho\}$. By the induction hypothesis, $\text{SUBTREE}([\![\tau]\!]_\eta, p_{\sigma'}) = [\![\mathsf{T}_{\sigma'}]\!]_{\eta_{\sigma'}}$, so by definition, σ' must have the form $nl(\ell = \vec{\alpha_\ell}.\sigma'_\ell)$. By the definitions, $\mathsf{T}_{\sigma'_\ell} = \tau_\ell\{f([p_{\sigma'}]) := \mathsf{T}_{\sigma'}\}$. Since $p_{\sigma_\ell} = p_\sigma\ell$ and $p_{\sigma'_\ell} = p_{\sigma'}\ell$, by the definition of equivalence of subtrees, $p_{\sigma'_\ell} \in [p_{\sigma_\ell}]$. By (2), all of this means that $\mathsf{T}_\sigma = \mathsf{rec}\ f([p_\sigma]).nl(\ell = \vec{\alpha_\ell}.\tau_\ell)$ and $\tau_\ell\{f([p_\sigma]) := \mathsf{T}_\sigma\} = \mathsf{T}_{\sigma_\ell}$ (3).

By definition and Lemma 4.10, $[\![\mathsf{T}_\sigma]\!]_{\eta_\sigma} = \mathsf{nl}(nl, \ell = [\![\tau_\ell\{f([p_\sigma]) := \mathsf{T}_\sigma\}]\!]_{\text{SHIFT}(\eta_\sigma, \vec{\alpha_\ell})})$. By the induction hypothesis and definition, $\text{SUBTREE}([\![\tau]\!]_\eta, p_\sigma\ell) = [\![\tau_\ell\{f([p_\sigma]) := \mathsf{T}_\sigma\}]\!]_{\eta_{\sigma_\ell}}$. By definition and (3), $\text{SUBTREE}([\![\tau]\!]_\eta, p_{\sigma_\ell}) = [\![\mathsf{T}_{\sigma_\ell}]\!]_{\eta_{\sigma_\ell}}$, as required.

$\sigma = \mathsf{rec}\,\alpha.\sigma'$**:** By the induction hypothesis, $\text{SUBTREE}([\![\tau]\!]_\eta, p_\sigma) = [\![\mathsf{T}_\sigma]\!]_{\eta_\sigma}$ and $\text{BIND}([\![\tau]\!]_\eta, p_\sigma) = n_\sigma$. By (1), and since $p_\sigma = p_{\sigma'}$, $\eta_\sigma = \eta_{\sigma'}$, and $n_\sigma = n_{\sigma'}$, $\text{SUBTREE}([\![\tau]\!]_\eta, p_{\sigma'}) = [\![\mathsf{T}_{\sigma'}]\!]_{\eta_{\sigma'}}$ and $\text{BIND}([\![\tau]\!]_\eta, p_{\sigma'}) = n_{\sigma'}$. Recall that α is fresh so $\sigma\{\rho_\sigma\} = \mathsf{rec}\,\alpha.(\sigma'\{\rho_\sigma\})$. By definition, $[\![\sigma\{\rho_\sigma\}]\!]_{\eta_\sigma} = \text{fix}([\![\alpha.\sigma'\{\rho_\sigma\}]\!]_{\eta_\sigma})$. By the induction hypothesis, $[\![\mathsf{T}_\sigma]\!]_{\eta_\sigma} = \text{fix}([\![\alpha.\sigma'\{\rho_\sigma\}]\!]_{\eta_\sigma})$. By the definition of fixed point, $[\![\mathsf{T}_\sigma]\!]_{\eta_\sigma} = [\![\alpha.\sigma'\{\rho_\sigma\}]\!]_{\eta_\sigma}([\![\mathsf{T}_\sigma]\!]_{\eta_\sigma})$. By Lemma 4.10, $[\![\mathsf{T}_\sigma]\!]_{\eta_\sigma} = [\![\sigma'\{\rho_\sigma\}\{\alpha := \mathsf{T}_\sigma\}]\!]_{\eta_\sigma}$. Since $\mathsf{T}_\sigma = \mathsf{T}_{\sigma'}$, $p_{\sigma'} = p_\sigma$, $\eta_{\sigma'} = \eta_\sigma$, and $\rho_{\sigma'} = \rho_\sigma, \alpha := \mathsf{T}_\sigma$, $[\![\mathsf{T}_{\sigma'}]\!]_{\eta_{\sigma'}} = [\![\sigma'\{\rho_{\sigma'}\}]\!]_{\eta_{\sigma'}}$, as required.

The final step of the proof is to show by induction on σ a subterm of τ that $\vdash \mathsf{T}_\sigma = \sigma\{\rho_\sigma\}$; the desired result is $\vdash \mathsf{T}_\tau = \tau\{\rho_\tau\}$. One property is important for the proofs. If σ_1 is a subterm of σ_2 then by induction, $\eta_{\sigma_1} = \text{SHIFT}(\eta_{\sigma_2}, \vec{\alpha})$ where $\vec{\alpha}$ are binders between σ_2 and σ_1 and $|\vec{\alpha}| = n_{\sigma_1} - n_{\sigma_2}$ (4). Consider the cases for σ:

σ **is free variable of** τ**:** In this case $\sigma\{\rho_\sigma\} = \sigma$. So $\text{SUBTREE}([\![\tau]\!]_\eta, p_\sigma) = [\![\sigma\{\rho_\sigma\}]\!]_{\eta_\sigma} = \eta_\sigma(\sigma)$. By (4), $\eta_\sigma(\sigma) = \text{SHIFT}(\eta(\sigma), n_\sigma)$. Since η is distinguishing, $\eta(\sigma) = \mathsf{var}(n)$ and since $n_\sigma = \text{BIND}([\![\tau]\!]_\eta, p_\sigma)$, $\text{SUBTREE}([\![\tau]\!]_\eta, p_\sigma) = \mathsf{var}(n + \text{BIND}([\![\tau]\!]_\eta, p_\sigma))$. By Lemma 3.6, $\text{VAROF}([\![\tau]\!]_\eta, p_\sigma) = \mathsf{free}(n)$. Since σ is a free variable, it is easy to establishd that $[p_\sigma] \notin S_\sigma$. By definition, $\mathsf{T}_\sigma = \mathsf{rec}\ f([p_\sigma]).\beta$ where $\eta(\beta) = \mathsf{var}(n)$. Since η is distinguising, $\beta = \sigma$. The result follows by (eqroll).

σ **is a bound variable of** τ**:** In this case $\sigma\{\rho_\sigma\} = \sigma$. So $\text{SUBTREE}([\![\tau]\!]_\eta, p_\sigma) = [\![\sigma\{\rho_\sigma\}]\!]_{\eta_\sigma} = \eta_\sigma(\sigma)$. By (4), $\eta_\sigma(\sigma) = \mathsf{var}(n + n_\sigma - n_{\sigma'\ell})$ where σ is the nth variable bound on the ℓ edge of σ'. Since $n_\sigma = \text{BIND}([\![\tau]\!]_\eta, p_\sigma)$ and $n_{\sigma'\ell} = \text{BIND}([\![\tau]\!]_\eta, p_{\sigma'}\ell)$, $\eta_\sigma(\sigma) = \mathsf{var}(n + \text{BIND}([\![\tau]\!]_\eta, p_{\sigma'}\ell \to p_\sigma))$. By Lemma 3.6, $\text{VAROF}([\![\tau]\!]_\eta, p_\sigma) = \mathsf{bound}(p_{\sigma'}, \ell, n)$. By similar reasoning to the previous case, $\mathsf{T}_\sigma = \mathsf{rec}\ f([p_\sigma]).f([p_{\sigma'}], \ell)_n$. Since σ' has the form $nl(\ell = f([p_{\sigma'}], \ell).\sigma'_\ell, \ldots)$, it must be that $f([p_{\sigma'}], \ell)_n = \sigma$. The result follows by (eqroll).

σ **is a recursively bound variable of** τ**:** In this case σ is bound by say σ'. By an easy induction, $\sigma\{\rho_\sigma\} = \mathsf{T}_{\sigma'}$. Thus $\text{SUBTREE}([\![\tau]\!]_\eta, p_\sigma) = [\![\sigma\{\rho_\sigma\}]\!]_{\eta_\sigma} = [\![\mathsf{T}_{\sigma'}]\!]_{\eta_\sigma}$. By (4), $\eta_\sigma = \text{SHIFT}(\eta_{\sigma'}, \vec{\alpha},)$ where $\vec{\alpha}$ are binders σ' and σ, that is, $\vec{\alpha}$ are not free variables of σ'. It is easy to establish that $\text{FTV}(\mathsf{T}_{\sigma'}) = \text{FTV}(\sigma')$. By Lemma 4.9, $[\![\mathsf{T}_{\sigma'}]\!]_{\eta_\sigma} = \text{SHIFT}([\![\mathsf{T}_{\sigma'}]\!]_{\eta'_\sigma}, |\vec{\alpha}|)$. By (4) also, $|\vec{\alpha}| = n_\sigma - n_{\sigma'} = \text{BIND}([\![\tau]\!]_\eta, p_{\sigma'} \to p_\sigma)$. Thus, combining all of these facts, $\text{SUBTREE}([\![\tau]\!]_\eta, p_\sigma) = \text{SHIFT}(\text{SUBTREE}([\![\tau]\!]_\eta, p_{\sigma'}), \text{BIND}([\![\tau]\!]_\eta, p_{\sigma'} \to p_\sigma))$. By Lemma 3.9, $p_{\sigma'}\ \text{EQST}([\![\tau]\!]_\eta)\ p_\sigma$. By (2), $\mathsf{T}_\sigma = \mathsf{T}_{\sigma'}$. The result follows by (eqref).

$\sigma = nl(\ell = \vec{\alpha_\ell}.\sigma_\ell)$ **:** By the induction hypothesis, $\vdash \mathsf{T}_{\sigma_\ell} = \sigma_\ell\{\rho_{\sigma_\ell}\}$, so by definition, $\vdash \mathsf{T}_{\sigma_\ell} = \sigma_\ell\{\rho_\sigma\}$. By (eqnl), $\vdash nl(\ell = \vec{\alpha_\ell}.\mathsf{T}_{\sigma_\ell}) = nl(\ell = \vec{\alpha_\ell}.\sigma_\ell\{\rho_\sigma\})$. By (3), $\mathsf{T}_\sigma = \mathsf{rec}\ f([p_\sigma]).nl(\ell = \vec{\alpha_\ell}.\tau_\ell)$ where $\tau_\ell\{f([p_\sigma]) := \mathsf{T}_\sigma\} = \mathsf{T}_{\sigma_\ell}$. Since $f([p_\sigma])$ and $\vec{\alpha_\ell}$ are fresh, the result follows by (eqroll).

$\sigma = \mathsf{rec}\,\alpha.\sigma'$**:** By the induction hypothesis, $\vdash \mathsf{T}_{\sigma'} = \sigma'\{\rho_{\sigma'}\}$. By (1) and definition, $\vdash \mathsf{T}_\sigma = \sigma'\{\rho_\sigma\}\{\alpha := \mathsf{T}_\sigma\}$. By (eqroll), $\vdash \sigma = \sigma'\{\alpha := \sigma\}$. By Lemma 4.22, $\vdash \sigma\{\rho_\sigma\} = \sigma'\{\alpha := \sigma\}\{\rho_\sigma\}$. Since α is fresh, $\vdash \sigma\{\rho_\sigma\} = \sigma'\{\rho_\sigma\}\{\alpha := \sigma\{\rho_\sigma\}\}$. Since $\sigma' \downarrow \alpha$ and α is fresh, it is not hard to show that $\sigma'\{\rho_\sigma\} \downarrow \alpha$. The result follows by (equnq).

$\square$

Completeness follows easily from this last lemma.

Theorem 4.24 (Completeness) *If $[\![\tau_1]\!]_\eta = [\![\tau_2]\!]_\eta$ for a distinguishing η then $\vdash \tau_1 = \tau_2$.*

Proof: By Lemma 4.23, $\vdash \textsc{termof}_\eta([\![\tau_1]\!]_\eta) = \tau_1$ and $\vdash \textsc{termof}_\eta([\![\tau_2]\!]_\eta) = \tau_2$. Since $[\![\tau_1]\!]_\eta = [\![\tau_2]\!]_\eta$, $\textsc{termof}_\eta([\![\tau_1]\!]_\eta) = \textsc{termof}_\eta([\![\tau_2]\!]_\eta)$. The result follows by (eqsym) and (eqtrans). $\square$

5 Binding Tree Automata

The final step of my programme is to give a decision algorithm for term equality. The algorithm is in terms of a finite representation for trees, similar to the term automata of Kozen et al. The basic idea is that an automaton is given as input a path and gives as output the node at the end of that path. Automata are state machines, that is, each label in the path causes the automaton to transition from one state to another, starting with an initial state, and the state obtained at the end of the path determines the output. Rather than output the node in the form a tree does, that is, as an element of $NL + \mathbb{N}$, the automaton outputs either a node label, a free variable index, or a bound variable. A bound variable is specified as a state, label, and index; the idea being that the variable is bound by the most recent occurrence of the binder generated by the state, along the edge given by the label.

A few preliminaries are needed for the definition.

Definition 5.1 *If $\delta \in Q \times L \rightharpoonup Q$ then δ^* is its extension to L^*, specifically $\delta^*(q, \epsilon) = q$ and $\delta^*(q, p\ell) = \delta(\delta^*(q, p), \ell)$. If $\delta^*(q_1, p) = q_2$ then p is a path from q_1 to q_2; if in addition $\delta^*(q_1, p') \neq q$ for $p' \leq p$ then p is a q-less path from q_1 to q_2.*

Definition 5.2 *A binding tree automaton is a quadruple (Q, i, δ, sl) satisfying:*

- *Q is a finite set called the states of the automaton.*
- *$i \in Q$ is the initial state.*
- *$\delta \in Q \times L \rightharpoonup Q$ is the transition function.*
- *$sl \in Q \to NL + \mathbb{N} + Q \times L \times \mathbb{N}$ is the labelling or output function.*
- *$(q, \ell) \in \mathrm{dom}(\delta) \Leftrightarrow sl(q) = \mathrm{in}_1(nl) \wedge \ell \in \textsc{labels}(nl)$.*
- *$sl(q) = \mathrm{in}_3(q', \ell, n)$ implies that:*
  - *$sl(q') = \mathrm{in}_1(nl)$,*
  - *$0 \leq n < \textsc{bind}(nl, \ell)$, and*
  - *if p is a path from i to q then there are paths p_1 and p_2 such that $p = p_1 \ell p_2$, p_1 is a path from i to q', and p_2 is q'-less (from $\delta(q', \ell)$ to q).*

5.1 Automata to Trees

This section explains how an automaton generates a tree. Letting t represent this tree then intuitively: if $sl(\delta^*(i,p)) = \mathrm{in}_1(nl)$ then $t(p) = \mathrm{in}_1(p)$; if $sl(\delta^*(i,p)) = \mathrm{in}_2(n)$ then $\textsc{varof}(t,p) = \mathsf{free}(n)$; and if $sl(\delta^*(i,p)) = \mathrm{in}_3(q',\ell,n)$ then $\textsc{varof}(t,p) = \mathsf{bound}(p',\ell,n)$ where p' is the longest path from i to q' that is a prefix of p.

The formal definition extends the state space to include enough information to compute the De Bruijn indices for free and bound variables. An extended state is a triple consisting of a state of the automaton, the number of variables bound along the path so far (needed to determine free variables), and a function $f \in Q \to \mathbb{N}$, which gives the number of variables bound since the last occurrence of each state (need to determine bound variables). The transition function is lifted to extended states in such a way as to track the binding information, and the labeling function is lifted to extended states to use the binding information to generate nodes for trees. Then the tree of an automaton at a path is just the lifted labeling function of the lifted transition function on the path.

Definition 5.3 *The tree associated with an automaton is defined as follows:*

$$
\begin{array}{lcl}
\textsc{shift}(f, q := n) & = & \lambda q'.\left\{\begin{array}{ll} 0 & q' = q \\ f(q') + n & q' \neq q \end{array}\right. \\
\hat{\delta}((q,n,f),\ell) & = & (\delta(q,\ell), n+i, \textsc{shift}(f, q := i)) \\
 & & \textit{where } sl(q) = \mathrm{in}_1(nl) \wedge i = \textsc{bind}(nl,\ell) \\
\hat{sl}(q,n,f) & = & \left\{\begin{array}{ll} \mathrm{in}_1(nl) & sl(q) = \mathrm{in}_1(nl) \\ \mathrm{in}_2(n+i) & sl(q) = \mathrm{in}_2(i) \\ \mathrm{in}_2(f(q')+i) & sl(q) = \mathrm{in}_3(q',\ell,i) \end{array}\right. \\
\textsc{tree}((Q,i,\delta,sl), qnf) & = & \lambda p.\hat{sl}(\hat{\delta}^*(qnf,p)) \\
\hat{\imath} & = & (i, 0, \lambda q.0) \\
\textsc{tree}(Q,i,\delta,sl) & = & \textsc{tree}((Q,i,\delta,sl),\hat{\imath})
\end{array}
$$

An automaton does in fact map to a tree.

Lemma 5.1 *If ta is an automaton then $\textsc{tree}(ta) \in \mathit{Tree}$.*

Proof: The result follows straightforwardly from the definitions. □

Latter proofs need some basic properties about the tree of of an automaton. These properties formalise the above intuition.

Lemma 5.2 *If $ta = (Q,i,\delta,sl)$ is a tree automaton then:*

- *If $\hat{\delta}^*(\hat{\imath},p) = (q,n,f)$ then $\textsc{bind}(\textsc{tree}(ta),p) = n$.*
- *If $\delta^*(i,p_1) = q_1$, $\hat{\delta}^*(\hat{\imath},p_1\ell p_2) = (q_2,n,f)$, and p_2 is q_1-less (from $\delta(q_1,\ell)$ to q_2) then $f(q_1) = \textsc{bind}(\textsc{tree}(ta), p_1\ell \to p_1\ell p_2)$.*

- $\textsc{varof}(\textsc{tree}(ta), p) = \mathsf{free}(n)$ *if and only if* $\delta^*(i,p) = q$ *and* $sl(q) = \mathrm{in}_2(n)$ *for some* q.

- $\textsc{varof}(\textsc{tree}(ta), p) = \mathsf{bound}(p', \ell, n)$ *if and only if there exists* q *and* q' *such that* $\delta^*(i,p) = q$, $\delta^*(i,p') = q'$, $sl(q) = \mathrm{in}_3(q', \ell, n)$, *and* p'' *is* q'*-less from* $\delta(q', \ell)$ *to* q', *where* $p = p'\ell p''$.

Proof: The first two items follow by a straightforward induction on the paths concerned. For the last two items, if $\textsc{tree}(ta)(p) = \mathrm{in}_2(i)$ then by the definition of the tree of an automaton, $\delta^*(i,p) = q$ such that $q = \mathrm{in}_2(j)$ or $q = \mathrm{in}_3(q', \ell, j)$. In the first case and by the first item, $i = j + \textsc{bind}(\textsc{tree}(ta), p)$ so by definition $\textsc{varof}(\textsc{tree}(ta), p) = \mathsf{free}(j)$, as required. In the second case, by the definition of automata, there are paths p' and p'' such that $p = p'\ell p''$, $\delta^*(i,p') = q'$, and p'' is q'-less. By the second item, $i = j + \textsc{bind}(\textsc{tree}(ta), p'\ell \to p)$, as required. □

5.2 Automata Characterise Regular Trees

The trees expressible with automata are exactly the regular trees. This section proves this.

Theorem 5.3 *If* ta *is an automaton then* $\textsc{tree}(ta) \in \mathit{RegTree}$.

Proof: Let ta be (Q, i, δ, sl). Let $R = \{(p_1, p_2) \mid \delta^*(i,p_1) = \delta^*(i,p_2)\}$. The rest of the proof shows that R is a nonoverlapping equivalence of $\textsc{tree}(ta)$'s subtrees. First, note that $\mathrm{dom}(\textsc{tree}(ta)) = \{p \mid \delta^*(i,p) \text{ defined}\}$, so R is a relation on $\mathrm{dom}(\textsc{tree}(ta))$. Second, R is an equivalence relation because $=$ is an equivalence relation. Third, let $p_1\ R\ p_2$. By definition, $\hat{\delta}(\hat{\imath}, p_1) = (q, n_1, f_1)$ and $\hat{\delta}(\hat{\imath}, p_2) = (q, n_2, f_2)$. Case 1, $sl(q) = \mathrm{in}_1(nl)$: Then $\hat{sl}(q, n_j, f_j) = \mathrm{in}_1(nl)$, and if $\ell \in \textsc{labels}(nl)$ then $\delta^*(i, p_1\ell) = \delta(q,\ell) = \delta^*(i, p_2\ell)$, so $p_1\ell\ R\ p_2\ell$. Thus $\textsc{eqstprop}(p_1, R, p_2)$, as required. Case 2, $sl(q) = \mathrm{in}_2(n)$: Then $\hat{sl}(q, n_j, f_j) = \mathrm{in}_2(n + n_j)$. By Lemma 5.2, $\textsc{bind}(\textsc{tree}(ta), p_j) = n_j$, so by Lemma 3.4, $\textsc{varof}(\textsc{tree}(ta), p_j) = \mathsf{free}(n)$. Thus $\textsc{eqstprop}(p_1, R, p_2)$, as required. Case 3, $sl(q) = \mathrm{in}_3(q', \ell, n)$: Then $\hat{sl}(q, n_j, f_j) = \mathrm{in}_2(f_j(q') + n)$. By definition, there are p_1', p_1'', p_2', and p_2'' such that $p_j = p_j'\ell p_j''$, $\delta^*(i, p_j') = q'$, and p_j'' is q'-less. By Lemma 5.2, $f_j(q') = \textsc{bind}(\textsc{tree}(ta), p_j'\ell \to p_j)$, so by Lemma 3.4, $\textsc{varof}(\textsc{tree}(ta), p_j) = \mathsf{bound}(p_j', \ell, n)$. Since $\delta^*(p_j') = q'$, $p_1'\ R\ p_2'$ and $\textsc{eqstprop}(p_1, R, p_2)$, as required.

To show that R is nonoverlapping, let p_1, p_2, and p_3 be such that $\textsc{varof}(\textsc{tree}(ta), p_3) = \mathsf{bound}(p_1, \ell, n)$ and $p_1 < p_2 < p_3$. Then by Lemma 5.2, $\delta^*(i, p_1) = q$ and p_3' is q-less from $\delta(q, \ell)$ to $\delta^*(p_3)$, where $p_3 = p_1\ell p_3'$. In particular, this means that $\delta^*(p_2) \neq q$, hence $(p_1, p_2) \notin R$ as required.

Since R can have at most $|Q|$ equivalence classes, which is finite, $\textsc{tree}(ta) \in \mathit{RegTree}$ as required. □

Theorem 5.4 *If* $t \in \mathit{RegTree}$ *then there exists an automaton* ta *such that* $\textsc{tree}(ta) = t$.

Proof: Let R be a nonoverlapping equivalence of $t's$ subtrees with a finite number of equivalence classes. Let $Q = [R]$, $i = [\epsilon]_R$, and $\delta = \lambda([p]_R, \ell).[p\ell]_R$; this is well defined by the conditions for being an equivalence of subtrees. Let sl be:

$$sl([p]_R) = \begin{cases} \mathrm{in}_1(nl) & t(p) = \mathrm{in}_1(nl) \\ \mathrm{in}_2(n) & \textsc{varof}(t,p) = \mathsf{free}(n) \\ \mathrm{in}_3([p']_R, \ell, n) & \textsc{varof}(t,p) = \mathsf{bound}(p', \ell, n) \end{cases}$$

This is well defined by the conditions for being an equivalence of subtrees. Let $ta = (Q, i, \delta, sl)$.

First, ta is a tree automaton. The first four conditions of the definition are immediate. For the fifth condition, $([p]_R, \ell) \in \mathrm{dom}(\delta)$ exactly when $[p\ell]_R$ is defined, that is, $p\ell \in \mathrm{dom}(t)$. By the definition of trees, this occurs exactly when $t(p) = \mathrm{in}_1(nl)$ and $\ell \in \textsc{labels}(nl)$. By the definition of sl, this occurs exactly when $sl([p]_R) = \mathrm{in}_1(nl)$ and $\ell \in \textsc{labels}(nl)$ as required. For the sixth condition, let p be such that $sl([p]_R) = \mathrm{in}_3(q', \ell, n)$. By the definition of sl this means that $\textsc{varof}(t,p) = \mathsf{bound}(p', \ell, n)$ for some $p' \in q'$. By Lemma 3.4, $0 \le n < \textsc{bind}(nl, \ell)$ where $t(p') = \mathrm{in}_1(nl)$. By the definition of sl, $sl([p']_R) = \mathrm{in}_1(nl)$ and $0 \le n < \textsc{bind}(nl,)$, thus satisfying the first two subconditions of the sixth subcondition. For the third subcondition, let p'' be a path from i to $[p]_R$, that is, $p'' \in [p]_R$. Then $\textsc{varof}(t, p'') = \mathsf{bound}(p''', \ell, n)$ for some $p''' \in q'$. Let p'''' be such that $p''' < p'''' < p''$. By the definition of nonoverlapping $(p''', p'''') \notin R$, so $p'''' \notin q'$. Thus the path from $\delta([p''']_R, \ell)$ to $[p'']_R$ is q'-less, as required.

Second, $\textsc{tree}(ta) = t$. It is easy to show by induction on p that $\textsc{bind}(t,p) = \textsc{bind}(\textsc{tree}(ta), p)$ $(*)$. Clearly, $\mathrm{dom}(\textsc{tree}(ta)) = \mathrm{dom}(\delta^*) = \mathrm{dom}(t)$; therefore, it remains to show that $t(p) = \textsc{tree}(ta)(p)$ for $p \in \mathrm{dom}(t)$. Let $p \in \mathrm{dom}(t)$, and $(q, n, f) = \hat{\delta}^*(\hat{\imath}, p)$. Clearly, $q = [p]_R$ and $\textsc{tree}(ta)(p) = \hat{sl}(q, n, f)$. Case 1, $t(p) = \mathrm{in}_1(nl)$: Then by definition $\hat{sl}(q,n,f) = \mathrm{in}_1(nl)$, as required. Case 2, $\textsc{varof}(t,p) = \mathsf{free}(i)$: By Lemma 3.4, $t(p) = \mathrm{in}_2(i + \textsc{bind}(t,p))$. By $(*)$ and Lemma 5.2, the latter is equal to $\mathrm{in}_2(i+n)$. By definition of sl, $\hat{sl}(q,n,f) = \mathrm{in}_2(i+n)$ as required. Case 3, $t(p) = \mathsf{bound}(p', \ell, i)$: By Lemma 3.4, $t(p) = \mathrm{in}_2(\textsc{bind}(t, p'\ell \to p) + i)$. By the definition of nonoverlapping, $(p', p'') \notin R$ for $p' < p'' < p$, which means that the path from $p'\ell$ to p is $[p']_R$-less. So by Lemma 5.2, $f([p']_R) = \textsc{bind}(\textsc{tree}(ta), p'\ell \to p)$. By definition, $\hat{sl}(q,n,f) = \mathrm{in}_2(\textsc{bind}(\textsc{tree}(ta), p'\ell \to p) + i)$. By $(*)$ the latter is equal to $\mathrm{in}_2(\textsc{bind}(t, p'\ell \to p) + i)$ as required. □

5.3 Equality Algorithm

Two trees are different if they differ at some path, but more specifically if they differ at some minimal path. This minimal path will be in the domain of both trees. Therefore to determine if two automata generate the different trees, it suffices to search for paths in their common domain that have different outputs. If the outputs along some path are the same up to but not including the last state then the number of variables bound up to the last state is the same. Thus, two free variable states will differ exactly when their indices differ, free variable states will differ from bound variable states, and bound variable states will differ if the most recent occurrence of the binding state occurred at different prefixes of the path. Thus determining if the outputs are different requires only keeping track of the states and the correspondence between binding states. As in Kozen et al., this can be expressed as a deterministic finite state automaton. The states of this equality automaton are triples, one state from each automata, and the correspondence between binding states. The transition function updates the states according to the input automata's transition functions and updates the correspondence. The accepting states are those where the output of the two states differs according to the binding state correspondence. The two trees differ if the language of the equality automaton is nonempty. The correspondence between binding states can be expressed as *partial bijections*, introduced next.

Definition 5.4 *The partial bijections between* A *and* B *are* $A \leftrightharpoons B = \{R \in \mathcal{P}(A \times B) \mid (a_1, b_1) \in R \wedge (a_2, b_2) \in R \Rightarrow (a_1 = a_2 \Leftrightarrow b_1 = b_2)\}$*. Bijection update of* R *by* a *maps to* b *is* $R\{a \leftrightharpoons b\} = \{(a', b') \in R \mid a' \neq a \wedge b' \neq b\} \cup \{(a, b)\}$.

Lemma 5.5 $\emptyset \in A \leftrightharpoons B$*. If* $R \in A \leftrightharpoons B$*,* $a \in A$*, and* $b \in B$ *then* $R\{a \leftrightharpoons b\} \in A \leftrightharpoons B$.

Proof: The results follow straightforwardly from the definitions. □

Definition 5.5 *The equality deterministic finite state automata (over alphabet* L*) of two automata* $ta_1 = (Q_1, i_1, \delta_1, sl_1)$ *and* $ta_2 = (Q_2, i_2, \delta_2, sl_2)$ *is:*

$$\begin{array}{lcl}
\textsc{correspond}_{sl_1,sl_2}(q_1, q_2, R) & = & \vee(sl_1(q_1) = \mathrm{in}_1(nl) \wedge sl_2(q_2) = \mathrm{in}_1(nl)) \\
 & & \vee(sl_1(q_1) = \mathrm{in}_2(n) \wedge sl_2(q_2) = \mathrm{in}_2(n)) \\
 & & \vee(sl_1(q_1) = \mathrm{in}_3(q_1', \ell, n) \wedge sl_2(q_2) = \mathrm{in}_3(q_2', \ell, n) \wedge q_1' \; R \; q_2') \\
\textsc{equal}(ta_1, ta_2) & = & (Q_1 \times Q_2 \times (Q_1 \leftrightharpoons Q_2), \\
 & & \;(i_1, i_2, \emptyset), \\
 & & \;\lambda(q_1, q_2, R, \ell).(\delta_1(q_1, \ell), \delta_2(q_2, \ell), R\{q_1 \leftrightharpoons q_2\}), \\
 & & \;\{(q_1, q_2, R) \mid \neg\textsc{correspond}_{sl_1,sl_2}(q_1, q_2, R)\})
\end{array}$$

Some properties of the equality automaton are required for proving the correctness of the algorithm. The last of these properties is the key part of the correctness argument.

Lemma 5.6 *If* $ta_1 = (Q_1, i_1, \delta_1, sl_1)$*,* $ta_2 = (Q_2, i_2, \delta_2, sl_2)$*, and* $\textsc{equal}(ta_1, ta_2) = (Q, i, \delta, F)$ *then:*

- $\mathrm{dom}(\delta^*(i)) = \mathrm{dom}(\delta_1^*(i_1)) \cap \mathrm{dom}(\delta_2^*(i_2)) = \mathrm{dom}(\textsc{tree}(ta_1)) \cap \mathrm{dom}(\textsc{tree}(ta_2))$
- *If* $\delta_1^*(i_1, p) = q_1$ *and* $\delta_2^*(i_2, p) = q_2$ *then* $\delta^*(i) = (q_1, q_2, f)$ *for some* f.
- *If* $\delta^*(i, p) = (q_1', q_2', R)$ *then:* $q_1 \; R \; q_2$ *if and only if there exists* p_1*,* ℓ*, and* p_2 *such that* $p = p_1 \ell p_2$*,* $\delta_1^*(i_1, p_1) = q_1$*,* $\delta_2^*(i_2, p_1) = q_2$*, and* p_2 *is* q_1*-less for* ta_1 *and* q_2*-less for* ta_2.
- *If* $\textsc{tree}(ta_1)(p') = \textsc{tree}(ta_2)(p')$ *for* $p' < p$ *then:* $\textsc{tree}(ta_1)(p) = \textsc{tree}(ta_2)(p)$ *if and only if* $\delta^*(i, p) \notin F$.

Proof: The first two items follow by a straightfoward induction of p. For the third item, by a straightforward induction, if $p = \ell_1 \cdots \ell_n$ and i, (q_1^1, q_1^2, R_1), $\ldots$, (q_n^1, q_n^2, R_n) are the corresponding states along the path, then $R = \emptyset\{q_1^1 \leftrightharpoons q_1^2\} \cdots \{q_{n-1}^1 \leftrightharpoons q_{n-1}^2\}$. Thus $q_1 \; R \; q_2$ implies that $q_1 = q_i^1$ and $q_2 = q_i^2$ for some i. Furthermore, the pair (q_i^1, q_i^2) will be deleted by pair (q_j^1, q_j^2) if $j > i$ and either $q_i^1 = q_j^1$ or $q_i^2 = q^2 j$. The result follows by straightforward reasoning from these facts.

For the fourth item, first note that $\text{TREE}(ta_1)(p') = \text{TREE}(ta_2)(p')$ for $p' < p$ implies that $\text{BIND}(\text{TREE}(ta_1), p) = \text{BIND}(\text{TREE}(ta_2), p)$ $(*)$. Let $\hat{\delta}_1^*(\hat{\imath}_1, p) = (q_1, n_1, f_1)$, $\hat{\delta}_2^*(\hat{\imath}_2, p) = (q_2, n_2, f_2)$, and $\delta^*(i, p) = (q_1, q_2, R)$. There are several cases. Case 1, $\text{TREE}(ta_1)(p) = \text{in}_1(nl_1)$ and $\text{TREE}(ta_2)(p) = \text{in}_1(nl_2)$: By the definition of $\text{TREE}(\cdot)$ it must be that $sl_1(q_1) = \text{in}_1(nl_1)$ and $sl_2(q_1) = \text{in}_1(nl_2)$. Then, $\text{TREE}(ta)(p) = \text{TREE}(ta)(p)$ exactly when $nl_1 = nl_2$, which is exactly when $\text{CORRESPOND}_{sl_1,sl_2}(q_1, q_2, R)$, that is, $\delta^*(i, p) \notin F$. Case 2, $\text{VAROF}(\text{TREE}(ta_1), p) = \mathsf{free}(i_1)$ and $\text{VAROF}(\text{TREE}(ta_1), p) = \mathsf{free}(i_2)$: By Lemma 5.2, $sl_1(q_1) = \text{in}_2(i_1)$ and $sl_2(q_2) = \text{in}_2(i_2)$. By $(*)$ and Lemma 3.4, $\text{TREE}(ta)(p) = \text{TREE}(ta)(p)$ exactly when $i_1 = i_2$, which is exactly when $\text{CORRESPOND}_{sl_1,sl_2}(q_1, q_2, R)$, that is, $\delta^*(i, p) \notin F$. Case 3, $\text{VAROF}(\text{TREE}(ta_1), p) = \mathsf{bound}(p_1, \ell_1, i_1)$ and $\text{VAROF}(\text{TREE}(ta_2), p) = \mathsf{bound}(p_2, \ell_2, i_2)$. Then by Lemma 5.2, $sl_1(q_1) = \text{in}_3(q_1', \ell_1, i_1)$, $sl_2(q_2) = \text{in}_3(q_2', \ell_2, i_2)$, p_1' is q_1'-less, and p_2' is q_2'-less, where $\delta_1^*(i_1, p_1) = q_1'$, $\delta_2^*(i_1, p_2) = q_2'$, $p = p_1\ell_1p_1'$, and $p = p_2\ell_2p_2'$. By $(*)$ and Lemma 3.4, $\text{TREE}(ta)(p) = \text{TREE}(ta)(p)$ exactly when $p_1 = p_2$ and $i_1 = i_2$. By the third item, the latter two are exactly when $q_1' \mathrel{R} q_2'$ and $i_1 = i_2$, which is exactly when $\text{CORRESPOND}_{sl_1,sl_2}(q_1, q_2, R)$ (note that $p_1 = p_2$ implies that $\ell_1 = \ell_2$), that is, when $\delta^*(i, p) \notin F$. Case 4, $\text{TREE}(ta_1)(p)$ is in the left summand and $\text{TREE}(ta_2)(p)$ is in the right summand (or vice versa): In this case $\text{TREE}(ta_1)(p) \neq \text{TREE}(ta_2)(p)$. By the definition of $\text{TREE}(\cdot)$, $sl_1(q_1)$ is in the first summand while $sl_2(q_2)$ is in the second or third summand (or vice versa). So $\neg\text{CORRESPOND}_{sl_1,sl_2}(q_1, q_2, R)$, that is, $\delta^*(i, P) \in F$. Case 5, $\text{VAROF}(\text{TREE}(ta_1), p)$ is a free variable and $\text{VAROF}(\text{TREE}(ta_2), p)$ is a bound variable (or vice versa): In this case because $(*)$ and Lemma 3.4, $\text{TREE}(ta_1)(p) \neq \text{TREE}(ta_2)(p)$. By Lemma 5.2, $sl_1(q_1)$ is in the second summand while $sl_2(q_2)$ is in the third summand (or vice versa). So $\neg\text{CORRESPOND}_{sl_1,sl_2}(q_1, q_2, R)$, that is, $\delta^*(i, P) \in F$. $\square$

The next theorem proves the correctness of the equality automaton.

Theorem 5.7 $\text{TREE}(ta_1) = \text{TREE}(ta_2) \Leftrightarrow L(\text{EQUAL}(ta_1, ta_2)) = \emptyset$

Proof: The result follows by straightfoward reasoning from Lemma 5.6, particularly the last item. $\square$

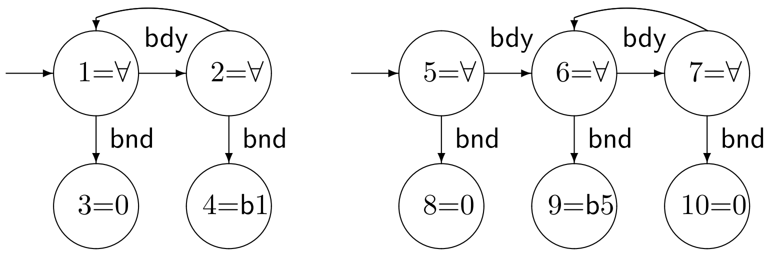
The previous theorem gives an algorithm for deciding the equality of the trees of two automata. Since emptiness of deterministic finite state automaton's language is linear time, and the equality automaton is exponential in the size of the tree automata, it is an exponential algorithm. However, an optimisation yields a polynomial time algorithm. This optimisation is based on the following lemma.

Lemma 5.8 *If* $ta_1 = (Q_1, i_1, \delta_1, sl_1)$ *and* $ta_2 = (Q_2, i_2, \delta_2, sl_2)$ *are automata,* $\delta_1^*(i_1, p_1) = \delta_1^*(i_1, p_1p_2) = \delta_1^*(i_1, p_1p_2p_3) = q_1$*,* $\delta_2^*(i_2, p_1) = \delta_2^*(i_2, p_1p_2) = \delta_2^*(i_2, p_1p_2p_3) = q_2$*,* $p_2 \neq \epsilon$*,* $p_3 \neq \epsilon$*, and* $p \notin L(\text{EQUAL}(ta_1, ta_2))$ *for* $p \ngeq p_1p_2p_3$ *then* $L(\text{EQUAL}(ta_1, ta_2)) = \emptyset$*.*

Proof: Let $ta_1 = (Q_1, i_1, \delta_1, sl_1)$ and $ta_2 = (Q_2, i_2, \delta_2, sl_2)$ be automata and let $(Q, i, \delta, F) = \text{EQUAL}(ta_1, ta_2)$. Let p_1, p_2, p_3, q_1, and q_2 be as in the statement of the theorem. Clearly, $\delta^*(i, p_1p_3) = (q_1, q_2, R_1)$, $\delta^*(i, p_1p_2) = (q_1, q_2, R_2)$, and $\delta^*(i, p_1p_2p_3) = (q_1, q_2, R_3)$ for some R_1, R_2, and R_3. The result follows if for all p, $\delta^*(i, p_1p_2p_3p) \notin F$. By a simple induction argument this reduces to showing that $\delta^*(i, p_1p_2p_3p) \notin F$ for all p such that p_3p and p_2p are not suffices of p_2p_3, so that $\delta^*(i, p_1p_3p), \delta^*(i, p_1p_2p) \notin F$ by the lemma's assumption. Clearly,

$\delta^*(i, p_1p_3p) = (q_1', q_2', R_1')$, $\delta^*(i, p_1p_2p) = (q_1', q_2', R_2')$, and $\delta^*(i, p_1p_2p_3p) = (q_1', q_2', R_3')$ for some q_1', q_2', R_1', R_2', and R_3'. If $sl_1(q_1')$ or $sl_2(q_2')$ is not in the third arm of the sum then $(q_1', q_2', R_3') \notin F$ because $(q_1', q_2', R_1') \notin F$ by assumption. If $sl_1(q_1') = \mathrm{in}_3(q_1'', \ell, n)$ then since $(q_1', q_2', R_1') \notin F$ and $(q_1', q_2', R_2') \notin F$, $sl_2(q_2') = \mathrm{in}_3(q_2'', \ell, n)$ for some q_2'' such that $q_1''\ R_1'\ q_2''$ and $q_1''\ R_2'\ q_2''$. In this case $(q_1', q_3', R_3') \notin F$ if $q_1''\ R_3'\ q_2''$. By Lemma 5.6, both p_1p_3p and p_1p_2p decompose into a path to (q_1'', q_2'') and then a (q_1'', q_2'')-less path. Case 1, $p_3 = p_3'\ell p_3''$, $\delta^*(i, p_1p_3') = (q_1'', q_2'', \_)$, and $p_3''p$ is (q_1'', q_2'')-less: Then $\delta^*(i, p_1p_2p_3') = (q_1'', q_2'', \_)$ and $p_3''p$ is (q_1'', q_2'')-less, so by Lemma 5.6, $q_1''\ R_3'\ q_2''$, as required. Case 2, $p_1p_2 = p_1'\ell p_2'$, $\delta^*(i, p_1') = (q_1'', q_2'', \_)$, and p_3p and $p_2'p$ are (q_1'', q_2'')-less: Then $p_2'p_3p$ is (q_1'', q_2'')-less, so by Lemma 5.6, $q_1''\ R_3'\ q_2''$, as required. □

The previous lemma says that the search for a word in $L(\text{EQUAL}(ta_1, ta_2))$ does not need to search the entire space $Q_1 \times Q_2 \times (Q_1 \leftrightharpoons Q_2)$, but only needs to search until it sees the same $Q_1 \times Q_2$ pair three times. It is insufficient to stop after seeing the same pair twice, as in Colazzo and Ghelli's algorithm. For example, consider the following two automata:[5]



The path bdybdybdy produces the pair $(2, 6)$ for the second time, and while bdybnd is not accepted by the equality automaton, bdybdybdybnd is accepted. The problem is that states 4 and 9 are part of repetitive structures and the path bdybnd checks that the first repetition of 4 matches the first repetition of 9, but not that the other repetitions match. The path bdybdybdybnd checks whether the second repetitions match. Only when both the first and second repetitions match will all repetitions necessarily match.

With an appropriate choice of representation and implemented carefully, the above algorithm is quadratic in the sizes of the input automata.

6 Putting it Together

The previous section gave an algorithm for deciding equality of two automata, so an algorithm to convert terms into automata gives an algorithm for deciding term equality. This section gives that algorithm and its correctness.

First an algorithm to convert terms into automata.

[5]The notation for automata depicts states as circles with $x = y$ inside; x is the state's identifier, and y is the states label, either a node label, a natural number for a free variable, or a bn for a variable bound by state n. An arrow not from another circle points to the initial state. The transition function is depicted with arrows between the circles labeled by L.

Definition 6.1 *If η is distinguishing then*

$$\textsc{automatonof}_\eta(\tau) = (\textsc{proper}(\tau), \textsc{proper}(\tau, \tau), \delta, sl)$$

where δ and sl are as follows. If σ is a subterm of τ and $\sigma = nl(\ell = \vec{\alpha_\ell}.\sigma_\ell)$ then $\delta(\sigma, \ell) = \textsc{proper}(\tau, \sigma_\ell)$ and $sl(\sigma) = \text{in}_1(nl)$. If σ is a subterm of τ that is an occurance of a free variable of τ then $sl(\sigma) = \text{in}_2(n)$ where $\eta(\sigma) = \mathsf{var}(n)$. If σ is subterm of τ that is an occurance of a variable bound by the nth binder of edge ℓ of σ' subterm of τ then $sl(\sigma) = \text{in}_3(\sigma', \ell, n)$.

A small technical result is required to prove the results of this section.

Lemma 6.1 *If* $\textsc{automatonof}_\eta(\tau) = (Q, i, \delta, sl)$ *then:* $p \in \textsc{paths}(\tau, q)$ *if and only if* $\delta^*(i, p) = q$.

Proof: The result follows straightforwardly by induction from the definitions. $\square$

The automaton of a term is in fact an automaton.

Lemma 6.2 *If η is distinguishing then* $\textsc{automatonof}_\eta(\tau)$ *is a tree automaton.*

Proof: Clearly the first four conditions of being a tree automaton are satisfied. Note that $(q, \ell) \in \text{dom}(\delta)$ only if q has the form $nl(\ell = \vec{\alpha_\ell}.\sigma_\ell)$ and $sl(q) = \text{in}_1(nl)$ only if q has that form. Thus $(q, \ell) \in \text{dom}(\delta)$ exactly when q has that form and when ℓ is one of the labels of the subterms, which is exactly when $\ell \in \textsc{labels}(nl)$ by syntactic constraints. Thus the fifth condition is satisfied. For the sixth condition, let $sl(q) = \text{in}_3(q', \ell, n)$ then q is a variable subterm of τ that is bound by the nth variable of the ℓ edge of subterm q' of ℓ. In this case q' must have the form $nl(\cdots)$, so $sl(q') = \text{in}_1(nl)$. Futhermore, because of the syntactic constraints on q', $\ell \in \textsc{labels}(nl)$ and $0 \le n < \textsc{bind}(nl, \ell)$. Finally, let p be a path from $\textsc{proper}(\tau, \tau)$ to q. By Lemma 6.1, $p \in \textsc{paths}(\tau, q)$. By Lemma 4.13, $p = p_1 \ell p_2$ such that $p_1 \in \textsc{paths}(\tau, q')$ and $p' \notin \textsc{paths}(\tau, q')$ for $p_1\ell \le p \le p_2$. By Lemma 6.1, p_1 is a path from $\textsc{proper}(\tau, \tau)$ to q' and p_2 is q'-less. $\square$

A term's automaton has the same tree as the term.

Theorem 6.3 *If η is distinguishing then* $\textsc{tree}(\textsc{automatonof}_\eta(\tau)) = [\![\tau]\!]_\eta$.

Proof: Let η be distinguishing and $\textsc{automatonof}_\eta(\tau) = (Q, i, \delta, sl)$. The result follows if $\textsc{tree}(\textsc{automatonof}_\eta(\tau))(p) = [\![\tau]\!]_\eta(p)$ for all $p \in \text{dom}(\textsc{tree}(\textsc{automatonof}_\eta(\tau)))$. The proof proceeds by induction on p. Let p be such that $\hat{\delta}^*(\hat{\imath}, p) = (q, n, f)$. By Lemma 6.1, $p \in \textsc{paths}(\tau, q)$. Case 1, $sl(q) = \text{in}_1(nl)$: In this case $\textsc{tree}(\textsc{automatonof}_\eta(\tau))(p) = \text{in}_1(nl)$. By definition, q has the form $nl(\cdots)$. By Lemma 4.17, $[\![\tau]\!]_\eta(p) = \text{in}_1(nl)$, as required. Case 2, $sl(q) = \text{in}_2(i)$: In this case $\textsc{tree}(\textsc{automatonof}_\eta(\tau))(p) = \text{in}_2(n + i)$. By Lemma 5.2, $n = \textsc{bind}(\textsc{tree}(\textsc{automatonof}_\eta(\tau)), p)$. By the induction hypothesis, the latter is $\textsc{bind}([\![\tau]\!]_\eta, p)$. By definition, $\eta(q) = \mathsf{var}(i)$. By Lemma 4.17, $\textsc{subtree}([\![\tau]\!]_\eta, p) = \textsc{shift}(\eta(q), \textsc{bind}([\![\tau]\!]_\eta, p)) = \textsc{shift}(\mathsf{var}(i), n)$. Thus $[\![\tau]\!]_\eta(p) = \text{in}_2(n + i)$, as required. Case 3, $sl(q) = \text{in}_3(q', \ell, i)$: In this case $sl(q) = \text{in}_2(f(q') + i)$. By definition of automata, $p = p_1 \ell p_2$ such that $\delta^*(i, p_1) = q'$ and p_2 is q'-less. By Lemma 5.2, $f(q') = \textsc{bind}(\textsc{tree}(\textsc{automatonof}_\eta(\tau)), p_1\ell \to p)$. By the induction hypothesis, $\textsc{bind}(\textsc{tree}(\textsc{automatonof}_\eta(\tau)), p_1\ell \to p) = \textsc{bind}([\![\tau]\!]_\eta, p_1\ell \to p)$. By definition, q is an occurance of a variable bound by the ith binder of edge ℓ of q'. By Lemma 6.1, $p_1 \in \textsc{paths}(\tau, q')$ and $p' \notin \textsc{paths}(\tau, q')$ for $p_1\ell \leq p' \leq p$. By Lemma 4.17, $[\![\tau]\!]_\eta(p) = \text{in}_2(i + \textsc{bind}([\![\tau]\!]_\eta, p_1\ell \to p))$, as required. $\square$

Finally, all the previous results can be combined into a decision procedure for equality of terms.

Theorem 6.4 *If η is distinguishing then:*

$$\vdash \tau_1 = \tau_2 \qquad \Leftrightarrow \qquad L(\textsc{equal}(\textsc{automatonof}_\eta(\tau_1), \textsc{automatonof}_\eta(\tau_2))) = \emptyset$$

Proof:

$$\begin{array}{ll}
 & \vdash \tau_1 = \tau_2 \\
\Leftrightarrow & \quad \langle\text{Theorem 4.20 and Theorem 4.24}\rangle \\
 & [\![\tau_1]\!]_\eta = [\![\tau_2]\!]_\eta \\
\Leftrightarrow & \quad \langle\text{Theorem 6.3}\rangle \\
 & \textsc{tree}(\textsc{automatonof}_\eta(\tau_1)) = \textsc{tree}(\textsc{automatonof}_\eta(\tau_2)) \\
\Leftrightarrow & \quad \langle\text{Theorem 5.7}\rangle \\
 & L(\textsc{equal}(\textsc{automatonof}_\eta(\tau_1), \textsc{automatonof}_\eta(\tau_2))) = \emptyset
\end{array}$$

$\square$

References

[AC93] Roberto Amadio and Luca Cardelli. Subtyping recursive types. *ACM Transactions on Progamming Languages and Systems*, 15(4):575–631, September 1993.

[Bru72] N. De Bruijn. Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation. *Indag. Mat.*, 34:381–392, 1972.

[CG99] Dario Colazzo and Giorgio Ghelli. Subtyping recursive types in kernel fun. In *1999 Symposium on Logic in Computer Science*, pages 137–146, Trento, Italy, July 1999.

[Gle02] Neal Glew. A theory of second-order trees. In *European Symposium on Programming 2002*, Grenoble, France, April 2002.

[Kap77] Irving Kaplansky. *Set Theory and Metric Spaces.* Chelsea Pub Co, 2nd edition, June 1977.

[KPS95] Dexter Kozen, Jens Palsberg, and Michael Schwartzbach. Efficient recursive subtyping. *Mathematical Structures in Computer Science*, 5(1):113–125, March 1995.