

Efficient Inference of Partial Types

Dexter Kozen* Jens Palsberg
kkozen@cs.cornell.edu palsberg@daimi.aau.dk

Michael I. Schwartzbach
mis@daimi.aau.dk

Computer Science Department, Aarhus University
Ny Munkegade, DK-8000 Aarhus C, Denmark

April 1992

Abstract

Partial types for the λ -calculus were introduced by Thatte in 1988 [3] as a means of typing objects that are not typable with simple types, such as heterogeneous lists and persistent data. In that paper he showed that type inference for partial types was semidecidable. Decidability remained open until quite recently, when O’Keefe and Wand [2] gave an exponential time algorithm for type inference.

In this paper we give an $O(n^3)$ algorithm. Our algorithm constructs a certain finite automaton that represents a canonical solution to a given set of type constraints. Moreover, the construction works equally well for recursive types; this solves an open problem stated in [2].

*Supported by the Danish Research Academy, the National Science Foundation, and the John Simon Guggenheim Foundation. On sabbatical from: Computer Science Department, Cornell University, Ithaca, New York 14853, USA.

1 Introduction

Partial types for the pure λ -calculus were introduced by Thatte in 1988 [3] as a way to type certain λ -terms that are untypable in the simply-typed λ -calculus. They are of substantial pragmatic value, since they allow the typing of such constructs as heterogeneous lists and persistent data that would otherwise be untypable.

Formally, partial types comprise a partially ordered set (T, \leq) , where T is the set of well-formed terms over the constant symbol Ω and the binary type constructor \rightarrow , and \leq is the partial order defined inductively as follows:

- (i) $t \leq \Omega$ for any t ;
- (ii) $s \rightarrow t \leq s' \rightarrow t'$ if and only if $s' \leq s$ and $t \leq t'$.

Intuitively, the type constructor \rightarrow represents the usual function space constructor, and Ω is a *universal type* that includes every other type. The partial order \leq can be thought of as *type inclusion* or *coercion*; that is, $s \leq t$ if it is possible to coerce type s into type t .

Clause (ii) in the definition of \leq models the fact that a function with domain s and range t can be coerced to a function with domain s' and range t' provided s' can be coerced to s and t can be coerced to t' ; thus the coercion order on functions is monotone in the range and antitone in the domain. That \leq is antitone in the domain is considered to be the main source of difficulty in type inference algorithms.

More λ -terms are typable with partial types than with simple types. For example, the term $\lambda f.(fK(fI))$, where $K = \lambda x.(\lambda y.x)$ and $I = \lambda z.z$, has partial type

$$(\Omega \rightarrow (\Omega \rightarrow \Omega)) \rightarrow \Omega,$$

but no simple type. The typing rules for the λ -calculus with partial types are the standard ones for subtypes [3].

As with any type discipline, the question of *type inference* is of paramount importance:

Given a λ -term E , is E typable? If so, give a type for it.

For this particular discipline, the type inference question can be rephrased in terms of solving a finite system of *type constraints*, which are just inequalities

over terms with type variables. Rephrased, the problem becomes:

Given a system of inequalities of the form $s \leq t$, where s and t are terms over \rightarrow and variables ranging over T , does the system have a solution in (T, \leq) ? If so, give a solution.

In his original paper [3], Thatte showed that the type inference problem for partial types is semidecidable. The problem of decidability remained unsolved until quite recently, when O’Keefe and Wand [2] presented an exponential time algorithm. Their algorithm involves iterated substitution and gives no hint of the possibility of the existence of canonical solutions; indeed, there exist satisfiable constraint systems with no \leq -minimal solution.

In this paper we show that the type inference problem for partial types is solvable in time $O(n^3)$, where n is the size of the λ -term. Moreover, the solutions we construct are *canonical* in the sense that they are least in the so-called *Böhm order*, a natural order different from \leq .

Our algorithm constructs a certain finite automaton with $O(n^2)$ states from the given system of type constraints. The canonical solution to the system is just the regular language accepted by the automaton, where we represent types as binary trees and binary trees as prefix-closed sets of strings over a two-letter alphabet. In this representation, the Böhm order is just set inclusion \subseteq .

The canonical solution always exists, but it may not be finite; however, since it is contained in all other solutions, we can check for the existence of a finite solution by checking whether the canonical solution is finite. Thus the typability question reduces essentially to the finiteness problem for regular sets.

Our construction works equally well for recursive types; this solves an open problem stated in [2].

Despite the fact that our polynomial-time algorithm now makes automatic type inference for partial types feasible, we feel that the more important contribution of this work is theoretical: namely, the precise mathematical characterization of the set of solutions to a system of type constraints and the identification of a canonical solution. We hope that the automata-theoretic approach developed here will be useful in dealing with other type systems.

2 From Types to Trees

In this section we rephrase the type inference problem and generalize to infinite trees. This allows us to isolate the essential combinatorial structure of the problem independent of type-theoretic syntax.

2.1 Systems of Type Constraints

Given a λ -term, the type inference question can be rephrased in terms of a system of type constraints as follows. Assume that the λ -term has been α -converted so that all bound variables are distinct. To each subterm E we assign a type variable $\llbracket E \rrbracket$. For every subterm of the form $\lambda x.E$ we generate the inequality $\llbracket \lambda x.E \rrbracket \geq \llbracket x \rrbracket \rightarrow \llbracket E \rrbracket$. For every subterm of the form EE' we generate the inequality $\llbracket E \rrbracket \leq \llbracket E' \rrbracket \rightarrow \llbracket EE' \rrbracket$. The solutions of this system of constraints correspond exactly to the possible types [3, 2].

2.2 Trees

Partial types are essentially binary trees. We represent binary trees as certain sets of strings over the binary alphabet $\{L, R\}$.

Definition 2.1 Let α, β, \dots denote elements of $\{L, R\}^*$. The *parity* of α is the number mod 2 of L 's in α . The parity of α is denoted $\pi\alpha$. A string α is said to be *even* (respectively, *odd*) if $\pi\alpha = 0$ (respectively, 1).

A *tree* is a subset $\sigma \subseteq \{L, R\}^*$ that is

- nonempty,
- closed under prefix, and
- binary, in the sense that for all $\alpha, \alpha R \in \sigma$ iff $\alpha L \in \sigma$.

We use σ, τ, \dots to denote trees. The set of all trees is denoted \widehat{T} .

A tree is *finite* if it is finite as a set of strings. A *path* in a tree σ is a maximal subset of σ linearly ordered by the prefix relation. By König's Lemma, a tree is finite iff it has no infinite paths.

An element $\alpha \in \sigma$ is a *leaf* of σ if it is not a proper prefix of any other element of σ . \square

For $A, B \subseteq \{L, R\}^*$ and $\alpha \in \{L, R\}^*$, define

$$\begin{aligned} A \cdot B &= \{\epsilon\} \cup \{L\alpha \mid \alpha \in A\} \cup \{R\beta \mid \beta \in B\} \\ A \downarrow \alpha &= \{\beta \mid \alpha\beta \in A\}. \end{aligned}$$

For trees σ, τ , $\sigma \cdot \tau$ is the tree with left subtree σ and right subtree τ , and $\sigma \downarrow \alpha$ is the subtree of σ at α if $\alpha \in \sigma$, \emptyset if not.

The following lemma establishes some elementary properties of the operators \cdot and \downarrow on trees.

Lemma 2.2

- (i) $(\sigma \cdot \tau) \downarrow L = \sigma$ and $(\sigma \cdot \tau) \downarrow R = \tau$
- (ii) $\sigma = \sigma \downarrow L \cdot \sigma \downarrow R$.
- (iii) $(\sigma \downarrow \alpha) \downarrow \beta = \sigma \downarrow \alpha\beta$.
- (iv) α is a leaf of σ iff $\sigma \downarrow \alpha = \{\epsilon\}$.

Proof. All properties are immediate consequences of the definitions. \square

The types T are in a natural one-to-one correspondence with the finite trees in \widehat{T} under the embedding $e : T \rightarrow \widehat{T}$ given by

$$\begin{aligned} e(\Omega) &= \{\epsilon\} \\ e(s \rightarrow t) &= e(s) \cdot e(t). \end{aligned}$$

Under this embedding, an occurrence of Ω at the fringe of a type s corresponds to a leaf of $e(s)$.

We now wish to define a partial order on \widehat{T} that agrees with the order \leq on T under the embedding e .

Definition 2.3 For $\sigma, \tau \in \widehat{T}$, define $\sigma \leq \tau$ if both of the following conditions hold for any α :

- (i) if α is an even leaf of σ , then $\alpha R \notin \tau$;
- (ii) if α is an odd leaf of τ , then $\alpha R \notin \sigma$.

\square

Lemma 2.4 *The relation \leq is a partial order on trees, and agrees with the order \leq on types under the embedding e . In particular, for any $\sigma, \tau, \sigma_i, \tau_i$,*

- (i) $\sigma \leq \{\epsilon\}$
- (ii) $\{\epsilon\} \leq \tau$ if and only if $\tau = \{\epsilon\}$;
- (iii) $\sigma_1 \cdot \sigma_2 \leq \tau_1 \cdot \tau_2$ if and only if $\tau_1 \leq \sigma_1$ and $\sigma_2 \leq \tau_2$.

Proof. We first show that \leq is a partial order. It is trivially reflexive. To show transitivity, let $\sigma \leq \tau \leq \omega$ and assume for a contradiction that α is an even leaf of σ and $\alpha R \in \omega$. Let β be the longest prefix of αR in τ . If $\beta = \alpha R$, then Definition 2.3(i) is violated for σ, τ . Otherwise β is a prefix of α and a leaf of τ . If β is even, then Definition 2.3(i) is violated for τ, ω . If β is odd, then Definition 2.3(ii) is violated for σ, τ . In all three cases we contradict the assumption $\sigma \leq \tau \leq \omega$. A symmetric argument under the assumption that α is an odd leaf of ω and $\alpha R \in \sigma$ likewise leads to a contradiction

For antisymmetry, assume $\sigma \leq \tau \leq \sigma$. Let $\alpha \in \sigma$ and let β be the longest prefix of α in τ . If $\beta \notin \alpha$, then β is a leaf of τ but then either β is even, which contradicts $\tau \leq \sigma$, or β is odd, which contradicts $\sigma \leq \tau$. Thus $\beta = \alpha$ and $\alpha \in \tau$. Since $\alpha \in \sigma$ was arbitrary, $\sigma \subseteq \tau$. A symmetric argument shows that $\tau \subseteq \sigma$.

We next establish the properties (i)—(iii) in turn.

(i) If α is an even leaf in σ , then clearly $\alpha R \notin \{\epsilon\}$. There are no odd leaves in $\{\epsilon\}$.

(ii) The *if* follows by reflexivity; *only if* follows by (i) and antisymmetry.

(iii) Let $\sigma = \sigma_1 \cdot \sigma_2$ and $\tau = \tau_1 \cdot \tau_2$. For *if*, assume that α is an even leaf in σ . We proceed by induction on the length of α . The case $\alpha = \epsilon$ is not possible. If $\alpha = L\beta$ then β is an odd leaf in σ_1 , so $\beta R \notin \tau_1$, so $L\beta R \notin \tau_1 \cdot \tau_2$, so $\alpha R \notin \tau$. If $\alpha = R\beta$ then β is an even leaf in σ_2 , so $\beta R \notin \tau_2$, so $R\beta R \notin \tau_1 \cdot \tau_2$, so $\alpha R \notin \tau$. Assume now that α is an odd leaf in τ . We proceed by induction on the length of α . The case $\alpha = \epsilon$ is not possible. If $\alpha = L\beta$ then β is an even leaf in τ_1 , so $\beta R \notin \sigma_1$, so $L\beta R \notin \sigma_1 \cdot \sigma_2$, so $\alpha R \notin \sigma$. If $\alpha = R\beta$ then β is an odd leaf in τ_2 , so $\beta R \notin \sigma_2$, so $R\beta R \notin \sigma_1 \cdot \sigma_2$, so $\alpha R \notin \sigma$.

For *only if*, assume that α is an even leaf in τ_1 ; then $L\alpha$ is an odd leaf in τ , so $L\alpha R \notin \sigma$, so $\alpha R \notin \sigma_1$. If α is an odd leaf in σ_1 , then $L\alpha$ is an even leaf in σ , so $L\alpha R \notin \tau$, so $\alpha R \notin \tau_1$. If α is an even leaf in σ_2 , then $R\alpha$ is an even

leaf in σ , so $R\alpha R \notin \tau$, so $\alpha R \notin \tau_2$. If α is an odd leaf in τ_2 , then $R\alpha$ is an odd leaf in τ , so $R\alpha R \notin \sigma$, so $\alpha R \notin \sigma_2$.

Finally, we show that the order on types agrees with the order on trees under the embedding e , *i.e.*, $s \leq t$ if and only if $e(s) \leq e(t)$. We proceed by induction on the structure of s and t . If $t = \Omega$ then the result follows from (i). If $s = \Omega$ then the result is immediate from (ii). If $s = s_1 \rightarrow s_2$ and $t = t_1 \rightarrow t_2$ then the induction hypothesis tells us that $t_1 \leq s_1$ if and only if $e(t_1) \leq e(s_1)$ and $s_2 \leq t_2$ if and only if $e(s_2) \leq e(t_2)$. The result now follows from (iii) and the definitions of e and \leq on types. \square

Amadio and Cardelli [1] give an alternative definition of a partial order on recursive types involving infinite chains of finite approximations. Definition 2.3 is equivalent to theirs.

Lemma 2.5 *The following properties hold for all σ, τ .*

$$(i) \quad (R \in \sigma \wedge R \in \tau \wedge \sigma \leq \tau) \Rightarrow (\sigma \downarrow L \geq \tau \downarrow L) \wedge (\sigma \downarrow R \leq \tau \downarrow R);$$

$$(ii) \quad (\sigma \leq \tau \wedge R \in \tau) \Rightarrow R \in \sigma.$$

Proof. Property (i) follows immediately from Lemma 2.4(iii) 7 and (ii) follows immediately from Lemma 2.4(ii). \square

3 From Constraints to Graphs

Instead of systems of type constraints involving type variables, we consider a more general notion of a *constraint graph*.

Definition 3.1 A *constraint graph* is a directed graph $G = (S, L, R, \leq)$ consisting of a set of nodes S and three sets of directed edges L, R, \leq . We write $s \xrightarrow{L} t$ to indicate that the pair (s, t) is in the edge set L , and similarly $s \xrightarrow{R} t$, $s \xrightarrow{\leq} t$. A constraint graph must satisfy the properties:

- any node has at most one outgoing L edge and at most one outgoing R edge;
- a node has an outgoing L edge if and only if it has an outgoing R edge.

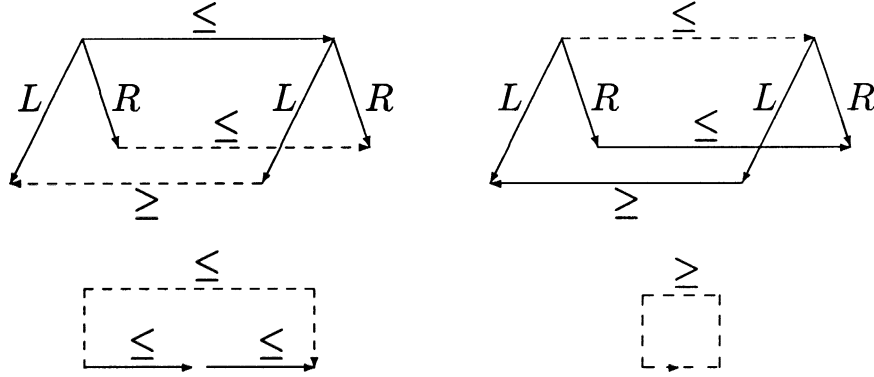
A *solution* for G is any map $h : S \rightarrow \widehat{T}$ such that

- (i) if $u \xrightarrow{L} v$ and $u \xrightarrow{R} w$, then $h(u) = h(v) \cdot h(w)$;
- (ii) if $u \xrightarrow{\leq} v$, then $h(u) \leq h(v)$.

The solution h is *finite* if $h(s)$ is a finite set for all s . □

A system of type constraints as described in §2.1 gives rise to a constraint graph by associating a unique node with every subterm, defining L and R edges from a term to its left and right subterms, and defining \leq edges for the inequalities.

Definition 3.2 A constraint graph is *closed* if whenever the following solid edges exist, then so do the dashed ones.



That is, the edge relation \leq is reflexive, transitive, and closed under rules resembling the two implications of Lemma 2.4(iii). The *closure* of a constraint graph G is the smallest closed graph containing G as a subgraph. □

Lemma 3.3 A constraint graph and its closure have the same set of solutions.

Proof. Any solution of the closure of G is also a solution of G , since G has fewer constraints. Conversely, the closure of G can be constructed from G by iterating the closure rules, and it follows inductively by Lemma 2.4 that any solution of G satisfies the additional constraints added by this process. □

4 From Graphs to Automata

In this section we define two automata \mathcal{M} and \mathcal{N} and describe their relationship. These automata will be used to characterize the canonical solution of a given constraint graph G . An intuitive account follows the formal definitions.

Definition 4.1 Let a closed constraint graph $G = (S, L, R, \leq)$ be given. The automaton \mathcal{M} is defined as follows. The input alphabet of \mathcal{M} is $\{L, R\}$. The states of \mathcal{M} are $S^2 \cup S^1 \cup S^0$. States in S^2 are written (s, t) , those in S^1 are written (s) , and the unique state in S_0 is written $()$. The transitions are defined as follows.

$$\begin{aligned}
 (u, v) &\xrightarrow{\epsilon} (u, v') && \text{if } v \leq v' \text{ in } G \\
 (u, v) &\xrightarrow{\epsilon} (u', v) && \text{if } u' \leq u \text{ in } G \\
 (u, v) &\xrightarrow{R} (u', v') && \text{if } u \xrightarrow{R} u' \text{ and } v \xrightarrow{R} v' \text{ in } G \\
 (u, v) &\xrightarrow{L} (v', u') && \text{if } u \xrightarrow{L} u' \text{ and } v \xrightarrow{L} v' \text{ in } G \\
 (u, v) &\xrightarrow{\epsilon} (v) && \text{always} \\
 (v) &\xrightarrow{\epsilon} (v') && \text{if } v \leq v' \text{ in } G \\
 (v) &\xrightarrow{R} (v') && \text{if } v \xrightarrow{R} v' \text{ in } G \\
 (v) &\xrightarrow{L} () && \text{if } v \xrightarrow{L} v' \text{ in } G
 \end{aligned}$$

If p and q are states of \mathcal{M} and $\alpha \in \{L, R\}^*$, we write $p \xrightarrow{\alpha} q$ if the automaton can move from state p to state q under input α , including possible ϵ -transitions.

The automaton \mathcal{M}_s is the automaton \mathcal{M} with start state (s, s) . All states are accept states; thus the language accepted by \mathcal{M}_s is the set of strings α for which there exists a state p such that $(s, s) \xrightarrow{\alpha} p$. We denote this language by $\mathcal{L}(s)$. \square

Informally, we can think of the automaton \mathcal{M}_s as follows. We start with two pebbles, one green and one red, on the node s of the constraint graph G . We can move the green pebble forward along a \leq edge at any time, and we can move the red pebble backward along a \leq edge at any time. We can move both pebbles simultaneously along R edges leading out of the nodes they occupy. We can also move them simultaneously along outgoing L edges, but in the latter case we switch their colors. At any time, we can elect to remove the red pebble; thereafter, we can move the green pebble forward

along \leq or R edges as often as we like, and forward along an L edge once, at which point the pebble must be removed. The sequence of L 's and R 's that were seen gives a string in $\mathcal{L}(s)$, and all strings in $\mathcal{L}(s)$ are obtained in this way.

The intuition motivating the definition of \mathcal{M} is that we want to identify the conditions that require a path to exist in any solution. Thus $\mathcal{L}(s)$ is the set of α that *must* be there; this intuition is made manifest in Lemma 4.2. It turns out that once we identify this set, we are able to show that it is a solution itself.

We now show that \mathcal{M} accepts only essential strings.

Lemma 4.2 *If $h : S \rightarrow \widehat{T}$ is any solution and $(s, s) \xrightarrow{\alpha} p$, then $\alpha \in h(s)$. Moreover,*

- (i) *if $p = (u, v)$ then $h(u) \leq h(s) \downarrow \alpha \leq h(v)$;*
- (ii) *if $p = (v)$ then $h(s) \downarrow \alpha \leq h(v)$;*

Proof. We proceed by induction on the number of transitions. If this is zero, then $p = (s, s)$ and $\alpha = \epsilon$, and the result is immediate. Otherwise, assume that $(s, s) \xrightarrow{\alpha} p$ and the lemma holds for this sequence of transitions. We argue by cases, depending on the form of the next transition out of p .

If p is of the form (u, v) , then the induction hypothesis says that $\alpha \in h(s)$ and $h(u) \leq h(s) \downarrow \alpha \leq h(v)$.

If $(u, v) \xrightarrow{\epsilon} (u', v')$, then $u' \xrightarrow{\leq} u$ and $v \xrightarrow{\leq} v'$ so $\alpha\epsilon = \alpha \in h(s)$ and

$$h(u') \leq h(u) \leq h(s) \downarrow \alpha \leq h(v) \leq h(v').$$

If $(u, v) \xrightarrow{R} (u', v')$, then $u \xrightarrow{R} u'$ and $v \xrightarrow{R} v'$, so $h(u') = h(u) \downarrow R$ and $h(v') = h(v) \downarrow R$. Then $R \in h(v)$, so by Lemmas 2.2 and 2.5, $R \in h(s) \downarrow \alpha$ and $\alpha R \in h(s)$ and

$$h(u') = h(u) \downarrow R \leq h(s) \downarrow \alpha R \leq h(v) \downarrow R = h(v').$$

If $(u, v) \xrightarrow{L} (v', u')$, then $u \xrightarrow{L} u'$ and $v \xrightarrow{L} v'$, so $h(u') = h(u) \downarrow L$ and $h(v') = h(v) \downarrow L$. Then $L \in h(v)$, so by Lemmas 2.2 and 2.5, $L \in h(s) \downarrow \alpha$ and $\alpha L \in h(s)$ and

$$h(u') = h(u) \downarrow L \geq h(s) \downarrow \alpha L \geq h(v) \downarrow L = h(v').$$

If $(u, v) \xrightarrow{\epsilon} (v)$, then $\alpha\epsilon = \alpha \in h(s)$ and $h(s) \downarrow \alpha \leq h(v)$.

If p is of the form (v) , then the induction hypothesis says that $\alpha \in h(s)$ and $h(s) \downarrow \alpha \leq h(v)$.

If $(v) \xrightarrow{\epsilon} (v')$, then $v \xrightarrow{\leq} v'$, so $\alpha\epsilon = \alpha \in h(s)$ and

$$h(s) \downarrow \alpha \leq h(v) \leq h(v').$$

If $(v) \xrightarrow{R} (v')$ then $(v) \xrightarrow{R} (v')$, so $h(v') = h(v) \downarrow R$. Then $R \in h(v)$ so by Lemmas 2.2 and 2.5, $R \in h(s) \downarrow \alpha$ and $\alpha R \in h(s)$, and

$$h(s) \downarrow \alpha R \leq h(v) \downarrow R = h(v')$$

Finally if $(v) \xrightarrow{L} (v')$, then $(v) \xrightarrow{L} v'$, so $h(v') = h(v) \downarrow L$. Then $L \in h(v)$, so by Lemmas 2.2 and 2.5, $L \in h(s) \downarrow \alpha$ and $\alpha L \in h(s)$. \square

Here we give a useful alternative characterization of $\mathcal{L}(s)$ in terms of a different automaton \mathcal{N} .

Definition 4.3 Let $G = (S, L, R, \leq)$ be given as above. We define the automaton \mathcal{N} over the input alphabet $\{L, R\}$ as follows. The states of \mathcal{N} are $S \times \{0, 1\}$; we use square brackets for states of \mathcal{N} to distinguish them from states of \mathcal{M} . The transitions are

$$\begin{aligned} [s, 0] &\xrightarrow{\epsilon} [t, 0] && \text{if } s \xrightarrow{\leq} t \text{ in } G \\ [s, 1] &\xrightarrow{\epsilon} [t, 1] && \text{if } t \xrightarrow{\leq} s \text{ in } G \\ [s, b] &\xrightarrow{R} [t, b] && \text{if } s \xrightarrow{R} t \text{ in } G \\ [s, b] &\xrightarrow{L} [t, \bar{b}] && \text{if } s \xrightarrow{L} t \text{ in } G \end{aligned}$$

As above, we write $[s, b] \xrightarrow{\alpha} [t, c]$ if $[s, b]$ can go to $[t, c]$ under α , including possible ϵ -transitions. \square

The automaton \mathcal{N} has states $[s, b]$ where b is a Boolean value. The second component is used to keep track of the parity of the spanned string. We can think of $[s, b]$ as a pebble on s ; the second component gives the color of the pebble. If the pebble is green ($b = 0$), we can move it forward along \leq edges. If the pebble is red ($b = 1$), we can move it backward along \leq edges. We can move the pebble forward along R or L edges at any time but if we move it along an L edges then we switch the color.

The following lemmas relate \mathcal{M} and \mathcal{N} .

Lemma 4.4

- (i) $(s, s) \xrightarrow{\alpha} (u, v)$ if and only if both
 - (a) $[s, \pi\alpha] \xrightarrow{\alpha} [v, 0]$
 - (b) $[s, \overline{\pi\alpha}] \xrightarrow{\alpha} [u, 1]$
- (ii) $(s) \xrightarrow{\alpha} (t)$ if and only if $\alpha = R^k$ for some k and $[s, 0] \xrightarrow{\alpha} [t, 0]$.
- (iii) $(s) \xrightarrow{\alpha} ()$ if and only if $\alpha = R^k L$ for some k and $[s, 0] \xrightarrow{\alpha} [t, 1]$ for some t .

Proof. We prove the three parts in turn. In each case we proceed by induction on α .

(i) If $\alpha = \epsilon$ then

$$\begin{aligned}
 (s, s) \xrightarrow{\epsilon} (u, v) &\Leftrightarrow u \lesssim s \wedge s \lesssim v \\
 &\Leftrightarrow [s, 0] \xrightarrow{\epsilon} [v, 0] \wedge [s, 1] \xrightarrow{\epsilon} [u, 1] \\
 &\Leftrightarrow [s, \pi\epsilon] \xrightarrow{\epsilon} [v, 0] \wedge [s, \overline{\pi\epsilon}] \xrightarrow{\epsilon} [u, 1]
 \end{aligned}$$

If $\alpha = \beta L$ then

$$(s, s) \xrightarrow{\beta} (p, q) \xrightarrow{L} (q', p') \xrightarrow{\epsilon} (u, v).$$

By the induction hypothesis, this is equivalent to

$$\begin{aligned}
 &[s, \pi\beta] \xrightarrow{\beta} [q, 0] \wedge [s, \overline{\pi\beta}] \xrightarrow{\beta} [p, 1] \wedge \\
 & p \xrightarrow{L} p' \wedge q \xrightarrow{L} q' \wedge u \lesssim q' \wedge p' \lesssim v \\
 \Leftrightarrow & [s, \pi\beta] \xrightarrow{\beta} [q, 0] \xrightarrow{L} [q', 1] \xrightarrow{\epsilon} [u, 1] \wedge \\
 & [s, \overline{\pi\beta}] \xrightarrow{\beta} [p, 1] \xrightarrow{L} [p', 0] \xrightarrow{\epsilon} [v, 0] \\
 \Leftrightarrow & [s, \overline{\pi\beta}] \xrightarrow{\beta L} [v, 0] \wedge [s, \pi\beta] \xrightarrow{\beta L} [u, 1] \\
 \Leftrightarrow & [s, \pi\beta L] \xrightarrow{\beta L} [v, 0] \wedge [s, \overline{\pi\beta L}] \xrightarrow{\beta L} [u, 1]
 \end{aligned}$$

If $\alpha = \beta R$ then

$$(s, s) \xrightarrow{\beta} (p, q) \xrightarrow{R} (p', q') \xrightarrow{\epsilon} (u, v).$$

By the induction hypothesis, this is equivalent to

$$[s, \pi\beta] \xrightarrow{\beta} [q, 0] \wedge [s, \overline{\pi\beta}] \xrightarrow{\beta} [p, 1] \wedge$$

$$\begin{aligned}
& p \xrightarrow{R} p' \wedge q \xrightarrow{R} q' \wedge u \xrightarrow{\leq} p' \wedge q' \xrightarrow{\leq} v \\
\Leftrightarrow & [s, \pi\beta] \xrightarrow{\beta} [q, 0] \xrightarrow{R} [q', 0] \xrightarrow{\epsilon} [v, 0] \wedge \\
& [s, \overline{\pi\beta}] \xrightarrow{\beta} [p, 1] \xrightarrow{R} [p', 1] \xrightarrow{\epsilon} [u, 1] \\
\Leftrightarrow & [s, \pi\beta] \xrightarrow{\beta R} [v, 0] \wedge [s, \overline{\pi\beta}] \xrightarrow{\beta R} [u, 1] \\
\Leftrightarrow & [s, \pi\beta R] \xrightarrow{\beta RL} [v, 0] \wedge [s, \overline{\pi\beta R}] \xrightarrow{\beta R} [u, 1]
\end{aligned}$$

(ii) If $\alpha = \epsilon$ then

$$\begin{aligned}
(s) \xrightarrow{\epsilon} (t) & \Leftrightarrow s \xrightarrow{\leq} t \\
& \Leftrightarrow \epsilon = R^0 \wedge [s, 0] \xrightarrow{\epsilon} [t, 0]
\end{aligned}$$

If $\alpha = \beta R$ then

$$(s) \xrightarrow{\beta} (p) \xrightarrow{R} (p') \xrightarrow{\epsilon} (t).$$

By the induction hypothesis, this is equivalent to

$$\begin{aligned}
\beta &= R^k \wedge [s, 0] \xrightarrow{\beta} [p, 0] \wedge p \xrightarrow{R} p' \wedge p' \xrightarrow{\leq} t \\
\Leftrightarrow & \beta R = R^{k+1} \wedge [s, 0] \xrightarrow{\beta R} [t, 0].
\end{aligned}$$

The case $\alpha = \beta L$ is not possible.

(iii) The cases $\alpha = \epsilon$ and $\alpha = \beta R$ are not possible. If $\alpha = \beta L$ then

$$(s) \xrightarrow{\beta} (p) \xrightarrow{L} ().$$

Using (ii), this is equivalent to

$$\begin{aligned}
\beta &= R^k \wedge [s, 0] \xrightarrow{\beta} [p, 0] \wedge p \xrightarrow{L} t \\
\Leftrightarrow & \alpha = R^k L \wedge [s, 0] \xrightarrow{\beta L} [t, 1].
\end{aligned}$$

□

Lemma 4.5 For any string $\alpha, \alpha \in \mathcal{L}(s)$ if and only if there exist β, k, u, v such that

- (i) $\alpha = \beta R^k$ or $\alpha = \beta R^k L$,
- (ii) $[s, \pi\beta] \xrightarrow{\alpha} [v, \pi\alpha \oplus \pi\beta]$, and
- (iii) $[s, \overline{\pi\beta}] \xrightarrow{\beta} [u, 1]$.

Here \oplus denotes addition mod 2.

Proof. First assume $\alpha \in \mathcal{L}(s)$. Then $(s, s) \xrightarrow{\alpha} p$ for some state p . If $p = (u, v)$ then $(s, s) \xrightarrow{\alpha} (u, v)$, so by Lemma 4.4 we have

$$[s, \pi\alpha] \xrightarrow{\alpha} [v, 0] \wedge [s, \overline{\pi\alpha}] \xrightarrow{\alpha} [u, 1].$$

But then we can choose $\alpha = \beta$ and $k = 0$. If $p = (v)$ then for some β, γ we have

$$(s, s) \xrightarrow{\beta} (u, q) \xrightarrow{\epsilon} (q) \xrightarrow{\gamma} (v).$$

so by Lemma 4.4 we have

$$\gamma = R^k \wedge [s, \pi\beta] \xrightarrow{\beta} [q, 0] \xrightarrow{\gamma} [v, 0] \wedge [s, \overline{\pi\beta}] \xrightarrow{\beta} [u, 1].$$

Since $\pi\alpha = \pi\beta$, this is equivalent to

$$\alpha = \beta R^k \wedge [s, \pi\beta] \xrightarrow{\alpha} [v, \pi\alpha \oplus \pi\beta] \wedge [s, \overline{\pi\beta}] \xrightarrow{\beta} [u, 1]$$

and we are done. If $p = ()$ then for some β, γ we have

$$(s, s) \xrightarrow{\beta} (u, q) \xrightarrow{\epsilon} (q) \xrightarrow{\gamma} ().$$

so by Lemma 4.4 we have

$$\gamma = R^k L \wedge [s, \pi\beta] \xrightarrow{\beta} [q, 0] \xrightarrow{\gamma} [v, 1] \wedge [s, \overline{\pi\beta}] \xrightarrow{\beta} [u, 1].$$

Since $\pi\alpha \neq \pi\beta$, this is equivalent to

$$\alpha = \beta R^k L \wedge [s, \pi\beta] \xrightarrow{\alpha} [v, \pi\alpha \oplus \pi\beta] \wedge [s, \overline{\pi\beta}] \xrightarrow{\beta} [u, 1]$$

and we are done.

Conversely, assume (i)–(iii). We have $\alpha = \beta\gamma$ where $\gamma = R^k$ or $R^k L$, and

$$\begin{aligned} [s, \pi\beta] &\xrightarrow{\alpha} [v, \pi\gamma] \\ [s, \overline{\pi\beta}] &\xrightarrow{\beta} [u, 1]. \end{aligned}$$

We must have

$$[s, \pi\beta] \xrightarrow{\beta} [p, 0] \xrightarrow{\gamma} [v, \pi\gamma]$$

for some p . From Lemma 4.4 we have

$$(s, s) \xrightarrow{\beta} (u, p) \xrightarrow{\epsilon} (p)$$

and either $(p) \xrightarrow{\gamma} (v)$ or $(p) \xrightarrow{\gamma} ()$, depending on whether $\gamma = R^k$ or $R^k L$. In either case $\alpha \in \mathcal{L}(s)$. \square

5 Main Result

In this section we prove the main result: that $\mathcal{L}(s)$ gives the canonical solution of G .

Theorem 5.1 *The sets $\mathcal{L}(s)$ are trees, and the function $\mathcal{L} : S \rightarrow \widehat{T}$ is a solution of G . Moreover, if $h : S \rightarrow \widehat{T}$ is any other solution, then $\mathcal{L}(s) \subseteq h(s)$ for any s .*

Proof. We first show that $\mathcal{L}(s) \in \widehat{T}$. It is clearly non-empty, since $(s, s) \xrightarrow{\epsilon} (s, s)$; it is prefix closed by definition; and it is binary because G always has L and R edges in pairs.

In order to show that \mathcal{L} is a solution of G , we need to show

- (i) if $u \xrightarrow{L} v$ and $u \xrightarrow{R} w$, then $\mathcal{L}(u) = \mathcal{L}(v) \cdot \mathcal{L}(w)$;
- (ii) if $u \xrightarrow{\leq} v$ then $\mathcal{L}(u) \leq \mathcal{L}(v)$.

First, we show (i) as two inclusions. Assume that $\alpha \in \mathcal{L}(v) \cdot \mathcal{L}(w)$. We proceed by induction on α . If $\alpha = \epsilon$ then we are done, since $\epsilon \in \mathcal{L}(u)$. If $\alpha = \mathcal{L}\beta$ then $\beta \in \mathcal{L}(v)$, so $(v, v) \xrightarrow{\beta} p$ for some p . From

$$(u, u) \xrightarrow{L} (v, v) \xrightarrow{\beta} p$$

we conclude that $L\beta \in \mathcal{L}(u)$. If $\alpha = R\beta$ then $\beta \in \mathcal{L}(w)$, so $(w, w) \xrightarrow{\beta} p$ for some p . From

$$(u, u) \xrightarrow{R} (w, w) \xrightarrow{\beta} p$$

we conclude that $R\beta \in \mathcal{L}(u)$.

Assume that $\alpha \in \mathcal{L}(u)$. We proceed by induction on α . If $\alpha = \epsilon$ then we are done, since $\epsilon \in \mathcal{L}(v) \cdot \mathcal{L}(w)$. If $\alpha = L\beta$ then from Lemma 4.5 there exist γ, k, p , and q such that $\beta = \gamma R^k$ or $\beta = \gamma R^k L$ and

$$[u, \pi L\gamma] \xrightarrow{\alpha} [p, \pi\alpha \oplus \pi L\gamma] \wedge [u, \overline{\pi L\gamma}] \xrightarrow{L\gamma} [q, 1]$$

Since $[u, \pi L\gamma] \xrightarrow{L} [v, \pi\gamma]$, $[u, \overline{\pi L\gamma}] \xrightarrow{L} [v, \overline{\pi\gamma}]$, and $\pi\alpha \oplus \pi L\gamma = \pi\beta \oplus \pi\gamma$ it follows that

$$[v, \pi\gamma] \xrightarrow{\beta} [p, \pi\beta \oplus \pi\gamma] \wedge [v, \overline{\pi\gamma}] \xrightarrow{\gamma} [q, 1]$$

so $\beta \in \mathcal{L}(v)$ and $\alpha \in \mathcal{L}(v) \cdot \mathcal{L}(w)$. If $\alpha = R\beta$ then from Lemma 4.5 there exist γ, k, p , and q such that $\beta = \gamma R^k$ or $\beta = \gamma R^k L$ and

$$[u, \pi R\gamma] \xrightarrow{\alpha} [p, \pi\alpha \oplus \pi R\gamma] \wedge [u, \overline{\pi R\gamma}] \xrightarrow{R\gamma} [q, 1]$$

Since $[u, \pi R\gamma] \xrightarrow{R} [w, \pi\gamma]$, $[u, \overline{\pi R\gamma}] \xrightarrow{R} [w, \overline{\pi\gamma}]$, and $\pi\alpha \oplus \pi R\gamma = \pi\beta \oplus \pi\gamma$ it follows that

$$[w, \pi\gamma] \xrightarrow{\beta} [p, \pi\beta \oplus \pi\gamma] \wedge [w, \overline{\pi\gamma}] \xrightarrow{\gamma} [q, 1]$$

so $\beta \in \mathcal{L}(w)$ and $\alpha \in \mathcal{L}(v) \cdot \mathcal{L}(w)$.

Second, we show (ii). We need to show that for any u, v, α ,

- if $u \xrightarrow{\leq} v$, α even, $\alpha \in \mathcal{L}(u)$, and $\alpha R \in \mathcal{L}(v)$, then $\alpha R \in \mathcal{L}(u)$;
- if $u \xrightarrow{\leq} v$, α odd, $\alpha \in \mathcal{L}(v)$, and $\alpha R \in \mathcal{L}(u)$, then $\alpha R \in \mathcal{L}(v)$.

Using the characterization in terms of \mathcal{N} , these two cases can be rolled into one: it suffices to show for any s, t, α ,

- if $[s, \pi\alpha] \xrightarrow{\epsilon} [t, \pi\alpha]$, $\alpha \in \mathcal{L}(s)$, and $\alpha R \in \mathcal{L}(t)$, then $\alpha R \in \mathcal{L}(s)$.

By Lemma 4.5, we have

$$[s, \pi\beta] \xrightarrow{\alpha} [u, \pi\alpha \oplus \pi\beta] \tag{1}$$

$$[s, \overline{\pi\beta}] \xrightarrow{\beta} [v, 1] \tag{2}$$

$$[t, \pi\beta'] \xrightarrow{\alpha R} [u', \pi\alpha R \oplus \pi\beta'] \tag{3}$$

$$[t, \overline{\pi\beta'}] \xrightarrow{\beta'} [v', 1]$$

Since αR ends with R , we must have $\alpha R = \beta' R^k$ for some k and $\pi\alpha R = \pi\beta'$; thus from (3) and $[s, \pi\alpha] \xrightarrow{\epsilon} [t, \pi\alpha]$ we get

$$[s, \pi\alpha] \xrightarrow{\alpha R} [u', 0] \tag{4}$$

If $\pi\alpha = \pi\beta$, we use (4) and (2) to get

$$[s, \pi\beta] \xrightarrow{\alpha R} [u', \pi\alpha \oplus \pi\beta]$$

$$[s, \overline{\pi\beta}] \xrightarrow{\beta} [v, 1]$$

If $\pi\alpha \neq \pi\beta 1$, we use (4) and (1) to get

$$\begin{aligned} [s, \pi\alpha] &\xrightarrow{\alpha R} [u', \pi\alpha R \oplus \pi\alpha] \\ [s, \overline{\pi\alpha}] &\xrightarrow{\alpha} [u, 1] \end{aligned}$$

In either case we have $\alpha R \in \mathcal{L}(s)$ by Lemma 4.5.

To show that \mathcal{L} is minimal, we need to show that for any other solution $h : S \rightarrow \widehat{T}$, $\mathcal{L}(s) \subseteq h(s)$ for all s . This follows directly from Lemma 4.2. \square

Recursive types are just regular trees [1]. The canonical solution we have constructed, although possibly infinite, is a regular tree. Thus we have solved the type inference problem for recursive types left open in [2]. Specifically, given a λ -term, we construct the corresponding constraint graph and automaton \mathcal{M} . Every subterm corresponds to a node s in the constraint graph, and its Böhm-minimal type annotation is represented by the language $\mathcal{L}(s)$.

Note that the Böhm-minimal type of any typable λ -term trivially is Ω . What we compute is the unique minimal Church-style explicit type annotation. For simple types we have that types and type annotations are isomorphic. This is not so for partial types. For example, the Böhm-minimal type of $\lambda f.(fK(fI))$ is just Ω , but its Böhm-minimal type annotation which our algorithm computes, is

$$\lambda f : (\Omega \rightarrow (\Omega \rightarrow \Omega)).(f(\lambda x : \Omega.\lambda y : \Omega.x)(f(\lambda z : \Omega.z)))$$

In the following section we give an efficient decision procedure for the existence of a *finite* types

6 An algorithm

We have argued that the type inference problem studied in [3, 2] is equivalent to the following: given a finite constraint graph G , does G have a finite solution? Using the characterization of the previous section we can answer this question easily.

Theorem 6.1 *We can decide in time $O(n^3)$ whether a constraint graph of size n has a finite solution.*

Proof. By Theorem 5.1, there exists a finite solution if and only if the canonical solution is finite. To determine this, we need only check whether

any $\mathcal{L}(s)$ contains an infinite path, We first form the constraint graph, then close it; this gives a graph with n vertices and $O(n^2)$ edges. This can be done in time $O(n^3)$. We then form the automaton \mathcal{M} , which has $n^2 + n + 1$ states but only $O(n^3)$ transitions, at most $O(n)$ from each state. We then check for a cycle with at least one non- ϵ transition reachable from some (s, s) . This can be done in linear time in the size of the graph using depth-first search. The entire algorithm requires time $O(n^3)$. \square

A λ -term of size n yields a constraint graph with $O(n)$ nodes and $O(n)$ edges. This gives

Corollary 6.2 *The type inference problem for partial types is solvable in time $O(n^3)$.*

7 A Characterization of All Solutions

We have shown that any solution $h : S \rightarrow \hat{T}$ of a constraint graph $G = (S, L, R, \leq)$ contains the canonical solution \mathcal{L} in the sense that $\mathcal{L}(s) \subseteq h(s)$ for all $s \in S$. However, there certainly exist functions $h : S \rightarrow \hat{T}$ containing \mathcal{L} in this sense that are not solutions. In this section we give a precise characterization of the set of all solutions of G .

In the followings we write \mathcal{L}_G for \mathcal{L} to denote the dependence on G .

Theorem 7.1 *Let $G = (S, L, R, \leq)$ be a constraint graph. A function $h : S \rightarrow \hat{T}$ is a solution of G if and only if there exists a (possibly infinite) constraint graph $G' = (S', L', R', \leq')$ containing G as a subgraph such that $h = \mathcal{L}_{G'}$ on S .*

Proof. First we show that if G is a subgraph of G' , then $\mathcal{L}_{G'}$ gives a solution of G . Suppose $u, v, w \in S$, $u \xrightarrow{L} v$, and $u \xrightarrow{R} w$. Since G is a subgraph of G' , $u \xrightarrow{L'} v$ and $u \xrightarrow{R'} w$. By Theorem 5.1, $\mathcal{L}_{G'}$ is a solution of G' , therefore $\mathcal{L}_{G'}(u) = \mathcal{L}_{G'}(v) \cdot \mathcal{L}_{G'}(w)$. Similarly, if $u, v \in S$ and $u \xrightarrow{\leq} v$, then $u \xrightarrow{\leq'} v$, therefore $\mathcal{L}_{G'}(u) \leq \mathcal{L}_{G'}(v)$. The two conditions of Definition 3.1 are met.

Conversely, let $h : S \rightarrow \hat{T}$ be any solution of G . We construct a constraint graph G' from h containing G as a subgraph and show that h and $\mathcal{L}_{G'}$ agree

on S . Define

$$\begin{aligned}
S' &= S \cup \{(s, \alpha) \mid s \in S, \alpha \in h(s)\} \\
L' &= L \cup \{((s, \alpha), (s, \alpha L)) \mid s \in S, \alpha L \in h(s)\} \\
R' &= R \cup \{((s, \alpha), (s, \alpha R)) \mid s \in S, \alpha R \in h(s)\} \\
\leq' &= \leq \cup \{((s, \epsilon), s) \mid s \in S\} \cup \{(s, (s, \epsilon)) \mid s \in S\}.
\end{aligned}$$

The graph $G' = (S', L', R', \leq')$ is a constraint graph, since each node has an L' successor iff it has an R' successor, and L' and R' successors are unique.

We show now that h agrees with $\mathcal{L}_{G'}$ on S . If $\alpha \in h(s)$, then $(s, \alpha) \in S'$, and there is a path from (s, ϵ) to (s, α) in G' with label α . In the automaton \mathcal{M}' constructed from G' , $(s, s) \xrightarrow{\alpha} ((s, \alpha), (s, \alpha))$, thus $\alpha \in \mathcal{L}_{G'}(s)$.

To show the reverse inclusion, we extend h in a natural way to a solution h' of G' , and then appeal to Theorem 5.1 to conclude that h' contains $\mathcal{L}_{G'}$. Define

$$\begin{aligned}
h'(s) &= h(s) \\
h'(s, \alpha) &= h(s) \downarrow \alpha.
\end{aligned}$$

It remains to show that h' is a solution of G' . If $s, t, u \in S$, $s \xrightarrow{L} t$, and $s \xrightarrow{R} u$, then

$$h'(s) = h(s) = h(t) \cdot h(u) = h'(t) \cdot h'(u).$$

If $\alpha L, \alpha R \in h(s)$, then by Lemma 2.2,

$$\begin{aligned}
h'(s, \alpha) &= h(s) \downarrow \alpha \\
&= (h(s) \downarrow \alpha) \downarrow L \cdot (h(s) \downarrow \alpha) \downarrow R \\
&= h(s) \downarrow \alpha L \cdot h(s) \downarrow \alpha R \\
&= h'(s, \alpha L) \cdot h'(s, \alpha R).
\end{aligned}$$

Finally, if $s, t \in S$ and $s \xrightarrow{\leq} t$, then

$$h'(s) = h(s) \leq h(t) = h'(t)$$

and to satisfy the inequalities $s \leq' (s, \epsilon) \leq' s$ we have

$$h(s) = h(s) \downarrow \epsilon = h'(s, \epsilon).$$

□

References

- [1] Ruberto M. Amadio and Luca Cardelli. Subtyping recursive types. In *Eighteenth Symposium on Principles of Programming Languages*. ACM Press, January 1991.
- [2] Patrick M. O’Keefe and Mitchell Wand, Type inference for partial types is decidable, In *Proc. ESOP’92, European Symposium on Programming*. Springer-Verlag (*LNCS* 582), 1992.
- [3] Satish Thatte. Type inference with partial types. In *Proc. International Colloquium on Automata, Languages, and Programming 1988*. Springer-Verlag (*LNCS* 317), 1988.