Type Inference with Non-structural Subtyping

Jens Palsberg* Mitchell Wand†‡ Patrick O'Keefe§

April 11, 1995

Abstract

We present an $O(n^3)$ time type inference algorithm for a type system with a largest type $\top$, a smallest type $\perp$, and the usual ordering between function types. The algorithm infers type annotations of minimal size, and it works equally well for recursive types. For the problem of typability, our algorithm is simpler than the one of Kozen, Palsberg, and Schwartzbach for type inference *without* $\perp$. This may be surprising, especially because the system with $\perp$ is strictly more powerful.

*Computer Science Department, Aarhus University, Ny Munkegade, DK–8000 Aarhus C., Denmark. E-mail: `palsberg@daimi.aau.dk`.

†Work supported by the National Science Foundation under grants CCR-9304144 and CCR-9404646.

‡College of Computer Science, Northeastern University, 360 Huntington Avenue, 161CN, Boston, MA 02115, USA. E-mail: `wand@ccs.neu.edu`.

§151 Coolidge Avenue #211, Watertown, MA 02172, USA. E-mail: `pmo@world.std.com`.

1 Introduction

This paper concerns types for the λ-calculus that can be generated from the grammar:

$$t ::= \bot \mid \top \mid t_1 \to t_2 \ .$$

Intuitively, $\bot$ is a least type containing only the divergent computation; $\top$ is a maximal or universal type containing all values; and $t_1 \to t_2$ is the usual function space.

Types are partially ordered by $\leq$ which is the smallest binary relation on types such that

1. $\bot \leq t \leq \top$ for all types t; and
2. $s \to t \leq s' \to t'$ if and only if $s' \leq s$ and $t \leq t'$.

Following [9], we denote this type system by PTB (PTB indicates *Partial Types with Bottom*). Thatte's system of partial types [7] did not include $\bot$; the fragment of PTB without $\bot$ will be denoted PT [9].

The type system PTB can be extended with recursive types, yielding Amadio and Cardelli's system [1], here denoted PTB_μ. The further extension of PTB_μ with the type Int of integers will be denoted $\mathsf{PTB}_\mu \cup \{\mathsf{Int}\}$.

We have earlier shown that PTB is strictly more powerful than PT [9], and recursive types add further power.

It is known that type inference for PT is computable in $O(n^3)$ time [3]. It is also known that type inference for $\mathsf{PTB}_\mu \cup \{\mathsf{Int}\}$ is computable in $O(n^3)$ time, by reduction to a flow analysis problem [5].

In this paper we show that type inference for PTB and PTB_μ is also computable in $O(n^3)$ time, by an algorithm similar to that in [3]. The algorithm infers type annotations of minimal size. A type is a tree, so its size is can be measured by the size of the set of paths from the root. In general, there can be several types of minimal size. For example, the λ-term $\lambda x.\lambda y.(\lambda f.f(fx))(\lambda v.vy)$ can be typed such that

$$\begin{array}{lcl} x & : & \bot \\ y & : & \bot \\ f & : & \bot \to \bot \\ v & : & \bot \ . \end{array}$$

These type annotations are of minimal size. If we replace the type of y by $\top$, then we obtain another type annotation of minimal size.

In the type system PT, every typable λ-term has exactly one type annotation of minimal size. The algorithm by Kozen, Palsberg, and Schwartzbach [3] infers this minimal-size type annotation.

The type inference algorithm by Palsberg and O'Keefe [5] for $\mathsf{PTB}_\mu \cup \{\mathsf{Int}\}$ does in general not infer type annotations of minimal size. It remains open if our result can be extended to $\mathsf{PTB}_\mu \cup \{\mathsf{Int}\}$.

For the problem of typability, our algorithm is simpler than the one of Kozen, Palsberg, and Schwartzbach for type inference *without* $\perp$. This may be surprising, especially because the system with $\perp$ is strictly more powerful. In our view, the simpler algorithm motivates having $\top$ and $\perp$ together in a type system.

In the following Section we define the type inference problem and summarize some known results that will be helpful in the later sections. In Section 3 we define an automaton that will be used when defining possible solutions to the type inference problem, and in Section 4 we define a useful notion of upper bound. In Section 5 we prove our main theorem that characterizes a minimal-size solution, and in Section 6 we present our type inference algorithm. Finally, in Section 7 we give an example of how our type inference algorithm works.

2 The type system

2.1 Types

Let $\Sigma = \{\rightarrow, \perp, \top\}$ be the ranked alphabet where $\rightarrow$ is binary and $\perp, \top$ are nullary. A *type* is a finite tree over Σ. A *path* from the root of such a tree is a string over $\{0, 1\}$, where 0 indicates "left subtree" and 1 indicates "right subtree". This set of types is the same as the one defined by the grammar in Section 1. We represent a type by a *term*, which is a function mapping each path from the root of the type to the symbol at the end of the path, as follows.

Definition 1 A *term* is a partial function

$$t : \{0, 1\}\star \rightarrow \Sigma$$

with domain $\mathcal{D}(t)$ such that $\mathcal{D}(t)$ is non-empty and prefix-closed, and such that if $t(\alpha) = \rightarrow$, then $\{i \mid \alpha i \in \mathcal{D}(t)\} = \{0,1\}$; and if $t(\alpha) \in \{\perp, \top\}$, then $\{i \mid \alpha i \in \mathcal{D}(t)\} = \emptyset$. The set of all such terms is denoted T_Σ. The set of terms with finite domain is denoted F_Σ. They represent finite trees, by König's Lemma. In our application, F_Σ is the set of types.

For $s, t \in T_\Sigma$, define $\rightarrow^{T_\Sigma}: T_\Sigma \times T_\Sigma \rightarrow T_\Sigma$ and $\perp^{T_\Sigma}, \top^{T_\Sigma} \in T_\Sigma$ by

$$
\begin{aligned}
(s \rightarrow^{T_\Sigma} t)(0\alpha) &= s(\alpha) \\
(s \rightarrow^{T_\Sigma} t)(1\alpha) &= t(\alpha) \\
(s \rightarrow^{T_\Sigma} t)(\epsilon) &= \rightarrow \\
\perp^{T_\Sigma}(\epsilon) &= \perp \\
\top^{T_\Sigma}(\epsilon) &= \top \ .
\end{aligned}
$$

Then

$$
\begin{aligned}
\mathcal{D}(s \rightarrow^{T_\Sigma} t) &= \{\epsilon\} \cup \{0\alpha \mid \alpha \in \mathcal{D}(s)\} \cup \{1\alpha \mid \alpha \in \mathcal{D}(t)\} \\
\mathcal{D}(\perp^{T_\Sigma}) &= \{\epsilon\} \\
\mathcal{D}(\top^{T_\Sigma}) &= \{\epsilon\} \ .
\end{aligned}
$$

For $t \in T_\Sigma$ and $\alpha \in \{0,1\}\star$, define the partial function $t \downarrow \alpha : \{0,1\}\star \rightarrow \Sigma$ by

$$
t \downarrow \alpha(\beta) \ = \ t(\alpha\beta) \ .
$$

If $t \downarrow \alpha$ has non-empty domain, then it is a term, and is called the subterm of t at position α.

For $x \subseteq \{0,1\}\star$ and $\alpha \in \{0,1\}\star$, define $x \downarrow \alpha = \{\beta \mid \alpha\beta \in x\}$. $\square$

The following properties are immediate from the definitions:

(i) $(s \rightarrow^{T_\Sigma} t) \downarrow 0 = s$

(ii) $(s \rightarrow^{T_\Sigma} t) \downarrow 1 = t$

(iii) $(t \downarrow \alpha) \downarrow \beta = t \downarrow \alpha\beta$

Following [2], we omit the superscript T_Σ on the operators $\rightarrow^{T_\Sigma}$, $\perp^{T_\Sigma}$, $\top^{T_\Sigma}$.

We will say that a term s has smaller size than a term t if $\mathcal{D}(s) \subseteq \mathcal{D}(t)$. Our type inference algorithm will infer types that are of minimal size.

Terms are ordered by the subtype relation $\leq$, as follows.

Definition 2 The *parity* of $\alpha \in \{0,1\}\star$ is the number mod 2 of 0's in α. The parity of α is denoted $\pi\alpha$. A string α is said to be *even* if $\pi\alpha = 0$ and *odd* if $\pi\alpha = 1$. Let $\leq_0$ be the partial order on Σ given by

$$\bot \leq_0 \to \quad \text{and} \quad \to \leq_0 \top$$

and let $\leq_1$ be its reverse

$$\top \leq_1 \to \quad \text{and} \quad \to \leq_1 \bot\ .$$

For $s, t \in T_\Sigma$, define $s \leq t$ if $s(\alpha) \leq_{\pi\alpha} t(\alpha)$ for all $\alpha \in \mathcal{D}(s) \cap \mathcal{D}(t)$. □

Kozen, Palsberg, and Schwartzbach [2] showed that the relation $\leq$ is equivalent to the order defined by Amadio and Cardelli [1]. The relation $\leq$ is a partial order with $\bot \leq t \leq \top$ for all $t \in T_\Sigma$, and $s \to t \leq s' \to t'$ if and only if $s' \leq s$ and $t \leq t'$ [1, 2]. Moreover, on F_Σ, $\leq$ coincides with the type ordering defined in Section 1 [2].

2.2 Type rules

If E is a λ-term, t is a type, and A is a type environment, *i.e.* a partial function assigning types to variables, then the judgement

$$A \vdash E : t$$

means that E has the type t in the environment A. Formally, this holds when the judgement is derivable using the following four rules:

$$A \vdash x : t \quad (\text{provided } A(x) = t) \tag{1}$$

$$\frac{A[x \leftarrow s] \vdash E : t}{A \vdash \lambda x.E : s \to t} \tag{2}$$

$$\frac{A \vdash E : s \to t \quad A \vdash F : s}{A \vdash EF : t} \tag{3}$$

$$\frac{A \vdash E : s \quad s \leq t}{A \vdash E : t} \tag{4}$$

The first three rules are the usual rules for simple types and the last rule is the rule of *subsumption*.

The type system has the subject reduction property, that is, if $A \vdash E : t$ is derivable and E β-reduces to E', then $A \vdash E' : t$ is derivable. This is proved by straightforward induction on the structure of the derivation of $A \vdash E : t$ [4].

This system types terms which are not typable in the simply-typed λ-calculus. For example, consider $\lambda f.(fK(fI))$, where K and I are the usual combinators. This is not typable in the ordinary calculus, since K and I have different types, but it is typable under partial typing: assign f the type $\top \to \top \to \top$. Both the K and I can be coerced to type $\top$, and the result (fI), of type $\top \to \top$, can be coerced to $\top$ to form the second argument of the first f. Therefore the entire term has type $(\top \to \top \to \top) \to \top$.

Similarly, some self-application is possible: $(\lambda x.xx)$ has type $(\top \to t) \to t$ for all t, since the final x can be coerced to $\top$.

However, not all terms are typable in this system. Any λ-term typable in PTB is strongly normalizing [9]. To indicate the flavor of those strongly normalizing λ-terms which are not typable in this system, we have earlier [9] showed that $(\lambda x.xxx)(\lambda y.y)$ is not typable in PTB. We also showed that $(\lambda f.f(fx))(\lambda v.vy)$ is typable in PTB but *not* in PT. This demonstrates that $\bot$ adds power to a type system.

2.3 Constraints

Given a λ-term E, the type inference problem can be rephrased in terms of solving a system of type constraints. Assume that E has been α-converted so that all bound variables are distinct. Let X_E be the set of λ-variables x occurring in E, and let Y_E be a set of variables disjoint from X_E consisting of one variable $[\![F]\!]$ for each occurrence of a subterm F of E. (The notation $[\![F]\!]$ is ambiguous because there may be more than one occurrence of F in E. However, it will always be clear from context which occurrence is meant.) We generate the following system of inequalities over $X_E \cup Y_E$:

- for every occurrence in E of a subterm of the form $\lambda x.F$, the inequality
$$x \to [\![F]\!] \;\; \leq \;\; [\![\lambda x.F]\!] \; ;$$
- for every occurrence in E of a subterm of the form GH, the inequality
$$[\![G]\!] \;\; \leq \;\; [\![H]\!] \to [\![GH]\!] \; ;$$

- for every occurrence in E of a λ-variable x, the inequality

$$x \;\leq\; [\![x]\!] \; .$$

Denote by $T(E)$ the system of constraints generated from E in this fashion. For every λ-term E, let $\mathsf{Tmap}(E)$ be the set of total functions from $X_E \cup Y_E$ to T_Σ. The function $\psi \in \mathsf{Tmap}(E)$ is a *solution* of $T(E)$ if and only if it is a solution of each constraint in $T(E)$. Specifically, for $V, V', V'' \in X_E \cup Y_E$:

The constraint:	has solution ψ if:
$V \to V' \leq V''$	$\psi(V) \to \psi(V') \leq \psi(V'')$
$V \leq V' \to V''$	$\psi(V) \leq \psi(V') \to \psi(V'')$
$V \leq V'$	$\psi(V) \leq \psi(V')$

The solutions of $T(E)$ correspond to the possible type annotations of E in a sense made precise by Theorem 3.

Let A be a type environment assigning a type to each λ-variable occurring freely in E. If ψ is a function assigning a type to each variable in $X_E \cup Y_E$, we say that ψ *extends* A if A and ψ agree on the domain of A.

Theorem 3 *The judgement $A \vdash E : t$ is derivable if and only if there exists a solution ψ of $T(E)$ extending A such that $\psi([\![E]\!]) = t$. In particular, if E is closed, then E is typable with type t if and only if there exists a solution ψ of $T(E)$ such that $\psi([\![E]\!]) = t$.*

Proof. Similar to the proof of Theorem 2.1 in the journal version of [3], in outline as follows. Given a solution of the constraint system, it is straightforward to construct a derivation of $A \vdash M\colon t$. Conversely, observe that if $A \vdash M\colon t$ is derivable, then there exists a derivation of $A \vdash M\colon t$ such that each use of one of the ordinary rules is followed by exactly one use of the subsumption rule. The approach in for example [8, 6] then gives a set of inequalities of the desired form. $\square$

2.4 Graphs

In this section we reduce the problem of solving constraint systems to a problem of solving a certain kind of constraint graphs. The graphs yield a convenient setting for applying algorithms like transitive closure.

Definition 4 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 $s \xrightarrow{0} t$ to indicate that the pair (s,t) is in the edge set L, we write $s \xrightarrow{1} t$ to indicate that the pair (s,t) is in the edge set R, and we write $s \xrightarrow{\leq} t$ to indicate that the pair (s,t) is in the edge set $\leq$. 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 $h : S \to T_\Sigma$ such that

(i) if $u \xrightarrow{0} v$ and $u \xrightarrow{1} w$, then $h(u) = h(v) \to h(w)$;

(ii) if $u \xrightarrow{\leq} v$, then $h(u) \leq h(v)$.

Notice that we consider solutions in $S \to T_\Sigma$, not just in $S \to F_\Sigma$. The solution h is *finite* if $h(s)$ is a finite set for all s.

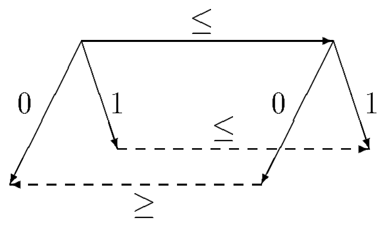
Define finally

$$N = \{u \in S \mid u \text{ has outgoing } L \text{ and } R \text{ edges in } G\}$$

$\square$

A system of type constraints as described above gives rise to a constraint graph by associating a unique node with every subexpression occurring in the system of constraints, defining L and R edges from an occurrence of an expression to its left and right subexpressions, and defining $\leq$ edges for the inequalities.

Definition 5 A constraint graph is *closed* if the edge relation $\leq$ is reflexive, transitive, and closed under the following rule which says that the dashed edges exist whenever the solid ones do:



The *closure* of a constraint graph G is the smallest closed graph containing G as a subgraph. □

Lemma 6 *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 that any solution of G satisfies the additional constraints added by this process. □

3 An automaton

In this section we define an automaton $\mathcal{M}$. It will be used to characterize a minimal-size assignment of types to nodes of a given constraint graph. An intuitive account follows the formal definition.

Definition 7 Let a constraint graph $G = (S, L, R, \leq)$ be given. The automaton $\mathcal{M}$ is defined as follows. The input alphabet of $\mathcal{M}$ is $\{0, 1\}$. The set of states of $\mathcal{M}$ is $S \times S$. A state is written (s, t). The transitions are defined as follows.

$$\begin{array}{ll} (u,v) \xrightarrow{\epsilon} (u,v') & \text{if } v \xrightarrow{\leq} v' \text{ in } G \\ (u,v) \xrightarrow{\epsilon} (u',v) & \text{if } u' \xrightarrow{\leq} u \text{ in } G \\ (u,v) \xrightarrow{1} (u',v') & \text{if } u \xrightarrow{1} u' \text{ and } v \xrightarrow{1} v' \text{ in } G \\ (u,v) \xrightarrow{0} (v',u') & \text{if } u \xrightarrow{0} u' \text{ and } v \xrightarrow{0} v' \text{ in } G \end{array}$$

If p and q are states of $\mathcal{M}$ and $\alpha \in \{0,1\}\star$, 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 (u, v) such that $(s, s) \xrightarrow{\alpha} (u, v)$. We denote this language by $\mathcal{L}(s)$. □

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. The sequence of 0's and 1's that were seen gives a string in $\mathcal{L}(s)$, and all strings in $\mathcal{L}(s)$ are obtained in this way.

For comparison, the automaton that was defined in [3] for doing type inference without $\perp$ had eight rather than four rules for generating transitions.

The intuition motivating the definition of $\mathcal{M}$ is that we want to identify the conditions that require a path to exist in the domain of any solution. Thus $\mathcal{L}(s)$ is the set of α that *must* be there; this intuition is made manifest in Lemma 8. It turns out that once we identify this set, we can construct a solution with domain $\mathcal{L}(s)$.

Lemma 8 *If $h : S \to T_\Sigma$ is any solution and $(s, s) \xrightarrow{\alpha} (u, v)$, then $\alpha \in \mathcal{D}(h(s))$. Moreover, $h(u) \leq h(s){\downarrow}\alpha \leq h(v)$.*

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

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

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

If $(u, v) \xrightarrow{1} (u', v')$, then $u \xrightarrow{1} u'$ and $v \xrightarrow{1} v'$, so $h(u') = h(u){\downarrow}1$ and $h(v') = h(v){\downarrow}1$. Then $1 \in \mathcal{D}(h(u))$ and $1 \in \mathcal{D}(h(v))$, so $1 \in \mathcal{D}(h(s){\downarrow}\alpha)$ and $\alpha 1 \in \mathcal{D}(h(s))$, and

$$h(u') \quad = \quad h(u){\downarrow}1 \quad \leq \quad h(s){\downarrow}\alpha 1 \quad \leq \quad h(v){\downarrow}1 \quad = \quad h(v') \ .$$

If $(u, v) \xrightarrow{0} (v', u')$, then $u \xrightarrow{0} u'$ and $v \xrightarrow{0} v'$, so $h(u') = h(u){\downarrow}0$ and $h(v') = h(v){\downarrow}0$. Then $0 \in \mathcal{D}(h(u))$ and $0 \in \mathcal{D}(h(v))$, so $0 \in \mathcal{D}(h(s){\downarrow}\alpha)$ and $\alpha 0 \in \mathcal{D}(h(s))$, and

$$h(u') \quad = \quad h(u){\downarrow}0 \quad \geq \quad h(s){\downarrow}\alpha 0 \quad \geq \quad h(v){\downarrow}0 \quad = \quad h(v') \ .$$

$\square$

When certain paths in $\mathcal{M}$ exist, others must exist too, as expressed in Lemmas 9 and 10.

Lemma 9 *If* $(u,v) \xrightarrow{\epsilon} (u',v')$*, then for any* $w \in S$*,* $(w,v) \xrightarrow{\epsilon} (w,v')$ *and* $(u,w) \xrightarrow{\epsilon} (u',w)$*.*

Proof. We proceed by induction on the number of transitions in $(u,v) \xrightarrow{\epsilon} (u',v')$. If it is zero, then $(u,v) = (u',v')$ and the result is immediate. Otherwise, assume that $(u,v) \xrightarrow{\epsilon} (u',v')$ and that the lemma holds for this sequence of transitions. We argue by cases, depending on the form of the next transition out of (u',v'). Clearly, the next transition need be an ϵ-transition.

If $(u',v') \xrightarrow{\epsilon} (u',v'')$ and $v' \stackrel{\leq}{\Rightarrow} v''$ in G, then, by the induction hypothesis,

$$(w,v) \xrightarrow{\epsilon} (w,v') \xrightarrow{\epsilon} (w,v'') \ .$$

If $(u',v') \xrightarrow{\epsilon} (u'',v')$ and $u'' \stackrel{\leq}{\Rightarrow} u'$ in G, then, by the induction hypothesis,

$$(u,w) \xrightarrow{\epsilon} (u',w) \xrightarrow{\epsilon} (u'',w) \ .$$

□

Lemma 10 *If* $(u_1,v_1) \xrightarrow{\alpha} (u_1',v_1')$ *and* $(u_2,v_2) \xrightarrow{\alpha} (u_2',v_2')$*, then* $(u_1,v_2) \xrightarrow{\alpha} (u_1',v_2')$*.*

Proof. We proceed by induction on the length of α. In the base case, consider $|\alpha| = 0$. We then have $\alpha = \epsilon$, so by Lemma 9,

$$(u_1,v_2) \xrightarrow{\epsilon} (u_1',v_2) \xrightarrow{\epsilon} (u_1',v_2') \ .$$

In the induction step, consider first

$$\begin{aligned} (u_1,v_1) &\xrightarrow{\alpha 1} (u_1',v_1') \\ (u_2,v_2) &\xrightarrow{\alpha 1} (u_2',v_2') \ . \end{aligned}$$

Choose $x_1,y_1,x_2,y_2 \in N$ and $x_1',y_1',x_2',y_2' \in S$ such that

$$\begin{aligned} (u_1,v_1) \xrightarrow{\alpha} (x_1,y_1) \xrightarrow{1} (x_1',y_1') \xrightarrow{\epsilon} (u_1',v_1') \\ (u_2,v_2) \xrightarrow{\alpha} (x_2,y_2) \xrightarrow{1} (x_2',y_2') \xrightarrow{\epsilon} (u_2',v_2') \end{aligned}$$

and $x_1 \xrightarrow{1} x_1'$, $y_1 \xrightarrow{1} y_1'$, $x_2 \xrightarrow{1} x_2'$, $y_2 \xrightarrow{1} y_2'$. From the induction hypothesis and Lemma 9 we get

$$(u_1, v_2) \xrightarrow{\alpha} (x_1, y_2) \xrightarrow{1} (x_1', y_2') \xrightarrow{\epsilon} (u_1', y_2') \xrightarrow{\epsilon} (u_1', v_2') \ .$$

Consider then

$$\begin{aligned}(u_1, v_1) &\xrightarrow{\alpha 0} (u_1', v_1') \\ (u_2, v_2) &\xrightarrow{\alpha 0} (u_2', v_2') \ .\end{aligned}$$

Choose $x_1, y_1, x_2, y_2 \in N$ and $x_1', y_1', x_2', y_2' \in S$ such that

$$\begin{aligned}(u_1, v_1) &\xrightarrow{\alpha} (x_1, y_1) \xrightarrow{0} (y_1', x_1') \xrightarrow{\epsilon} (u_1', v_1') \\ (u_2, v_2) &\xrightarrow{\alpha} (x_2, y_2) \xrightarrow{0} (y_2', x_2') \xrightarrow{\epsilon} (u_2', v_2')\end{aligned}$$

and $x_1 \xrightarrow{0} x_1'$, $y_1 \xrightarrow{0} y_1'$, $x_2 \xrightarrow{0} x_2'$, $y_2 \xrightarrow{0} y_2'$. From the induction hypothesis and Lemma 9 we get

$$(u_1, v_2) \xrightarrow{\alpha} (x_2, y_1) \xrightarrow{0} (y_1', x_2') \xrightarrow{\epsilon} (u_1', x_2') \xrightarrow{\epsilon} (u_1', v_2') \ .$$

□

The fundamental relation between a closed constraint graph G and the automaton $\mathcal{M}$ is expressed by Lemma 11.

Lemma 11 *Suppose G is a closed constraint graph. If $u \overset{\leq}{\Rightarrow} v$ and $(u, v) \xrightarrow{\alpha} (u', v')$, then $u' \overset{\leq}{\Rightarrow} v'$.*

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

If $(u', v') \xrightarrow{\epsilon} (u', v'')$ and $v' \overset{\leq}{\Rightarrow} v''$ in G, then, by the induction hypothesis, $u' \overset{\leq}{\Rightarrow} v'$, so since $\leq$ is transitive we have $u' \overset{\leq}{\Rightarrow} v''$.

If $(u', v') \xrightarrow{\epsilon} (u'', v')$ and $u'' \overset{\leq}{\Rightarrow} u'$ in G, then, by the induction hypothesis, $u' \overset{\leq}{\Rightarrow} v'$, so since $\leq$ is transitive we have $u'' \overset{\leq}{\Rightarrow} v'$.

If $(u', v') \xrightarrow{1} (u'', v'')$ and both $u' \xrightarrow{1} u''$ and $v' \xrightarrow{1} v''$, then, by the induction hypothesis, $u' \overset{\leq}{\Rightarrow} v'$, so since G is closed we have $u'' \overset{\leq}{\Rightarrow} v''$.

If $(u', v') \xrightarrow{0} (v'', u'')$ and both $u' \xrightarrow{0} u''$ and $v' \xrightarrow{0} v''$, then, by the induction hypothesis, $u' \overset{\leq}{\Rightarrow} v'$, so since G is closed we have $v'' \overset{\leq}{\Rightarrow} u''$. □

4 Upper bounds

We want to construct a solution $\psi : S \to T_\Sigma$ such that $\mathcal{D}(\psi(s)) = \mathcal{L}(s)$ for all $s \in S$. Suppose $\mathcal{L}(u) \downarrow \alpha = \{\epsilon\}$ for some $u \in S$. Then we want $(\psi(u))(\alpha) \in \{\bot, \top\}$. The question is: should we pick $\bot$ or $\top$? To answer this question, we consider $\mathsf{Up}(u, \alpha)$, which intuitively is the set of upper bounds of u at level α, and is defined as follows.

Definition 12 Define $\mathsf{Up} : S \times \{0,1\}^* \to 2^N$ as follows:

$$\mathsf{Up}(s, \alpha) = \{w \in N \mid (s, s) \xrightarrow{\alpha} (u, w) \text{ for some } u \in S\}$$

$\square$

Lemma 13 *Suppose $u \overset{\leq}{\Rightarrow} v$ and $\alpha \in \mathcal{L}(u) \cap \mathcal{L}(v)$.*

1. *If $\mathsf{Up}(u, \alpha) = \emptyset$, then $\mathsf{Up}(v, \alpha) = \emptyset$.*
2. *If $\mathsf{Up}(v, \alpha) = \emptyset$ and $\pi\alpha = 1$, then $\mathsf{Up}(u, \alpha) = \emptyset$.*
3. *If $\mathcal{L}(u) \downarrow \alpha \neq \{\epsilon\}$, $\mathcal{L}(v) \downarrow \alpha = \{\epsilon\}$, then $\mathsf{Up}(v, \alpha) = \emptyset$.*
4. *If $\mathcal{L}(v) \downarrow \alpha \neq \{\epsilon\}$, $\mathcal{L}(u) \downarrow \alpha = \{\epsilon\}$, and $\pi\alpha = 1$, then $\mathsf{Up}(u, \alpha) = \emptyset$.*

Proof. To prove (1), suppose $\mathsf{Up}(v, \alpha) \neq \emptyset$. Choose $w \in N$ and $u' \in S$ such that $(v, v) \xrightarrow{\alpha} (u', w)$. Choose also $u_1, v_1 \in S$ such that $(u, u) \xrightarrow{\alpha} (u_1, v_1)$. From $u \overset{\leq}{\Rightarrow} v$ and Lemma 10 we then get

$$(u, u) \xrightarrow{\epsilon} (u, v) \xrightarrow{\alpha} (u_1, w) ,$$

contradicting $\mathsf{Up}(u, \alpha) = \emptyset$.

To prove (2), suppose $\mathsf{Up}(u, \alpha) \neq \emptyset$. Since $\pi\alpha = 1$, we can write $\alpha = \beta 0 \gamma$ where $\beta \in \{0,1\}\star$ and $\gamma \in \{1\}\star$. Choose $x_1, y_1, w \in N$ and $x_2, y_2, u' \in S$ such that

$$(u, u) \xrightarrow{\beta} (x_1, y_1) \xrightarrow{0} (y_2, x_2) \xrightarrow{\gamma} (u', w)$$

and $x_1 \xrightarrow{0} x_2$ and $y_1 \xrightarrow{0} y_2$. Choose $p_1, q_1 \in N$ and $p_2, q_2, u'', v'' \in S$ such that

$$(v, v) \xrightarrow{\beta} (p_1, q_1) \xrightarrow{0} (q_2, p_2) \xrightarrow{\gamma} (u'', v'')$$

and $p_1 \xrightarrow{0} p_2$ and $q_1 \xrightarrow{0} q_2$. From $u \overset{\leq}{\Rightarrow} v$ and Lemma 10 we then get

$$(v,v) \xrightarrow{\epsilon} (u,v) \xrightarrow{\beta} (x_1,q_1) \xrightarrow{0} (q_2,x_2) \xrightarrow{\gamma} (u'',w) ,$$

contradicting $\mathsf{Up}(v,\alpha) = \emptyset$.

To prove (3), suppose $\mathsf{Up}(v,\alpha) \neq \emptyset$. Choose $w \in N$ and $u' \in S$ such that $(v,v) \xrightarrow{\alpha} (u',w)$. Choose also $x, y \in N$ such that $(u,u) \xrightarrow{\alpha} (x,y)$. From $u \overset{\leq}{\Rightarrow} v$ and Lemma 10 we get

$$(v,v) \xrightarrow{\epsilon} (u,v) \xrightarrow{\alpha} (x,w) ,$$

contradicting $\mathcal{L}(v)\downarrow\alpha = \{\epsilon\}$.

To prove (4), suppose $\mathsf{Up}(u,\alpha) \neq \emptyset$. Since $\pi\alpha = 1$, we can write $\alpha = \beta 0 \gamma$ where $\beta \in \{0,1\}\star$ and $\gamma \in \{1\}\star$. Choose $x_1, y_1, w \in N$ and $x_2, y_2, u' \in S$ such that

$$(u,u) \xrightarrow{\beta} (x_1,y_1) \xrightarrow{0} (y_2,x_2) \xrightarrow{\gamma} (u',w)$$

and $x_1 \xrightarrow{0} x_2$ and $y_1 \xrightarrow{0} y_2$. Since $\mathcal{L}(v)\downarrow\alpha \neq \{\epsilon\}$, choose $p_1, q_1, u'', v'' \in N$ and $p_2, q_2 \in S$ such that

$$(v,v) \xrightarrow{\beta} (p_1,q_1) \xrightarrow{0} (q_2,p_2) \xrightarrow{\gamma} (u'',v'')$$

and $p_1 \xrightarrow{0} p_2$ and $q_1 \xrightarrow{0} q_2$. From $u \overset{\leq}{\Rightarrow} v$ and Lemma 10 we then get

$$(u,u) \xrightarrow{\epsilon} (u,v) \xrightarrow{\beta} (x_1,q_1) \xrightarrow{0} (q_2,x_2) \xrightarrow{\gamma} (u'',w) ,$$

contradicting $\mathcal{L}(u)\downarrow\alpha = \emptyset$. □

Lemma 14 *Suppose $i \in \{0,1\}$, $u \xrightarrow{i} v$, and $\alpha \in \mathcal{L}(v)$.*

1. *If $\mathsf{Up}(u,i\alpha) = \emptyset$, then $\mathsf{Up}(v,\alpha) = \emptyset$.*

2. *Suppose G is closed. If $\mathsf{Up}(v,\alpha) = \emptyset$, then $\mathsf{Up}(u,i\alpha) = \emptyset$.*

Proof. To prove (1), suppose $\mathsf{Up}(v,\alpha) \neq \emptyset$. Choose $w \in N$ and $u' \in S$ such that

$$(u,u) \xrightarrow{i} (v,v) \xrightarrow{\alpha} (u',w) ,$$

contradicting $\mathsf{Up}(u, i\alpha) = \emptyset$.

To prove (2), suppose $\mathsf{Up}(u, i\alpha) \neq \emptyset$. Choose $w \in N$ and $u', u_1, v_1 \in S$ such that

$$(u, u) \xrightarrow{i} (u_1, v_1) \xrightarrow{\alpha} (u', w) \ .$$

From $(u, u) \xrightarrow{i} (v, v)$ and Lemma 10 we get

$$\begin{aligned}(u, u) &\xrightarrow{i} (u_1, v) \\ (u, u) &\xrightarrow{i} (v, v_1) \ .\end{aligned}$$

From $u \xrightarrow{\leq} u$ and Lemma 11 we get $u_1 \xrightarrow{\leq} v \xrightarrow{\leq} v_1$, hence

$$(v, v) \xrightarrow{\epsilon} (u_1, v_1) \xrightarrow{\alpha} (u', w) \ ,$$

contradicting $\mathsf{Up}(v, \alpha) = \emptyset$. □

Lemma 15 *Suppose* $i \in \{0, 1\}$, $u \xrightarrow{i} v$, *and* $\alpha \in \mathcal{L}(v)$. *If* G *is closed and* $\mathcal{L}(v) \downarrow \alpha = \{\epsilon\}$, *then* $\mathcal{L}(u) \downarrow i\alpha = \{\epsilon\}$.

Proof. Suppose $\mathcal{L}(u) \downarrow i\alpha \neq \{\epsilon\}$. Choose $x, y \in N$ and $u_1, v_1 \in S$ such that

$$(u, u) \xrightarrow{i} (u_1, v_1) \xrightarrow{\alpha} (x, y) \ .$$

From $(u, u) \xrightarrow{i} (v, v)$ and Lemma 10 we get

$$\begin{aligned}(u, u) &\xrightarrow{i} (u_1, v) \\ (u, u) &\xrightarrow{i} (v, v_1) \ .\end{aligned}$$

From $u \xrightarrow{\leq} u$ and Lemma 11 we get $u_1 \xrightarrow{\leq} v \xrightarrow{\leq} v_1$, hence

$$(v, v) \xrightarrow{\epsilon} (u_1, v_1) \xrightarrow{\alpha} (x, y) \ ,$$

contradicting $\mathcal{L}(v) \downarrow \alpha = \{\epsilon\}$. □

5 Main result

Definition 16 Let a closed constraint graph $G = (S, L, R, \leq)$ be given. Define $\psi : S \to T_\Sigma$ as follows:

$$\psi(s) = \lambda\alpha. \begin{cases} \top & \text{if } \alpha \in \mathcal{L}(s) \wedge \mathcal{L}(s)\downarrow\alpha = \{\epsilon\} \wedge \mathsf{Up}(s,\alpha) = \emptyset \\ \bot & \text{if } \alpha \in \mathcal{L}(s) \wedge \mathcal{L}(s)\downarrow\alpha = \{\epsilon\} \wedge \mathsf{Up}(s,\alpha) \neq \emptyset \\ \to & \text{if } \alpha \in \mathcal{L}(s) \wedge \mathcal{L}(s)\downarrow\alpha \neq \{\epsilon\} \\ \textit{undefined} & \text{if } \alpha \notin \mathcal{L}(s) \end{cases}$$

□

Theorem 17 *If G is a closed constraint graph, then the function ψ is a solution of G. Moreover, if $h : S \to T_\Sigma$ is any other solution of G, then $\mathcal{D}(\psi(s)) \subseteq \mathcal{D}(h(s))$ for any s.*

Proof. For any $s \in S$, notice that $\psi(s)$ is nonempty since $(s,s) \xrightarrow{\epsilon} (s,s)$, that it is prefix-closed since all states of $\mathcal{M}_s$ are accept states, that if $(\psi(s))(\alpha) =\to$ then $\{i \mid \alpha i \in \mathcal{D}(\psi(s))\} = \{0,1\}$, and that if $(\psi(s))(\alpha) \in \{\bot, \top\}$ then $\{i \mid \alpha i \in \mathcal{D}(\psi(s))\} = \emptyset$, so $\psi(s) \in T_\Sigma$. Notice also that $\mathcal{D}(\psi(s)) = \mathcal{L}(s)$ for all $s \in S$.

We first prove that

$$\text{If } u \xrightarrow{0} v_0 \text{ and } u \xrightarrow{1} v_1, \text{ then } \psi(u) = \psi(v_0) \to \psi(v_1).$$

Notice that $\psi(u) = \psi(v_0) \to \psi(v_1)$ is equivalent to

$$(\psi(u))(\alpha) = (\psi(v_0) \to \psi(v_1))(\alpha) \text{ for all } \alpha \in \mathcal{D}(\psi(u)) \cap \mathcal{D}(\psi(v_0) \to \psi(v_1)). \tag{5}$$

To see this equivalence, notice that (5) implies both $\psi(u) \leq \psi(v_0) \to \psi(v_1)$ and $\psi(u) \geq \psi(v_0) \to \psi(v_1)$.

To prove (5), notice first that $(\psi(u))(\epsilon) = (\psi(v_0) \to \psi(v_1))(\epsilon) =\to$. Then, let $i \in \{0,1\}$ and $i\alpha \in \mathcal{D}(\psi(u)) \cap \mathcal{D}(\psi(v_0) \to \psi(v_1))$. It is sufficient to prove

$$(\psi(u))(i\alpha) = (\psi(v_i))(\alpha) \ .$$

There are three cases. Suppose $(\psi(v_i))(\alpha) =\to$. Then $\mathcal{L}(v_i)\downarrow\alpha \neq \{\epsilon\}$. From $(u,u) \xrightarrow{i} (v_i, v_i)$ we get $\mathcal{L}(u)\downarrow i\alpha \neq \{\epsilon\}$, so $(\psi(u))(i\alpha) =\to$. Suppose then $(\psi(v_i))(\alpha) = \bot$. Then $\mathcal{L}(v_i)\downarrow\alpha = \{\epsilon\}$ and $\mathsf{Up}(v_i, \alpha) \neq \emptyset$. From

Lemma 14.1 we get $\mathsf{Up}(u, i\alpha) \neq \emptyset$. From Lemma 15 we get $\mathcal{L}(u) \downarrow i\alpha = \{\epsilon\}$, so $(\psi(u))(i\alpha) = \bot$. Suppose finally $(\psi(v_i))(\alpha) = \top$. Then $\mathcal{L}(v_i) \downarrow \alpha = \{\epsilon\}$ and $\mathsf{Up}(v_i, \alpha) = \emptyset$. From Lemma 14.2 we get $\mathsf{Up}(u, i\alpha) = \emptyset$. From Lemma 15 we get $\mathcal{L}(u) \downarrow i\alpha = \{\epsilon\}$, so $(\psi(u))(i\alpha) = \top$.

We then prove that

$$\text{If } u \overset{\leq}{\Rightarrow} v, \text{ then } \psi(u) \leq \psi(v).$$

Suppose $\alpha \in \mathcal{D}(\psi(u)) \cap \mathcal{D}(\psi(v)) = \mathcal{L}(u) \cap \mathcal{L}(v)$. We then need to prove $(\psi(u))(\alpha) \leq_{\pi\alpha} (\psi(v))(\alpha)$. There are two cases.

Suppose first $\pi\alpha = 0$. If $(\psi(u))(\alpha) = \bot$, then the result is immediate. If $(\psi(u))(\alpha) = \top$, then $\mathcal{L}(u) \downarrow \alpha = \{\epsilon\}$ and $\mathsf{Up}(u, \alpha) = \emptyset$. By Lemma 13.1, $\mathsf{Up}(v, \alpha) = \emptyset$. Hence, $\mathcal{L}(v) \downarrow \alpha = \{\epsilon\}$, so $(\psi(v))(\alpha) = \top$, from which the result follows. If $(\psi(u))(\alpha) = \rightarrow$, then there are two cases. If $\mathcal{L}(v) \downarrow \alpha \neq \{\epsilon\}$, then $(\psi(v))(\alpha) = \rightarrow$, from which the result follows. If $\mathcal{L}(v) \downarrow \alpha = \{\epsilon\}$, then it follows from Lemma 13.3 that $\mathsf{Up}(v, \alpha) = \emptyset$. Hence, $(\psi(v))(\alpha) = \top$, from which the result follows.

Suppose then $\pi\alpha = 1$. If $(\psi(v))(\alpha) = \bot$, then the result is immediate. If $(\psi(v))(\alpha) = \top$, then $\mathcal{L}(v) \downarrow \alpha = \{\epsilon\}$ and $\mathsf{Up}(v, \alpha) = \emptyset$. By Lemma 13.2, $\mathsf{Up}(u, \alpha) = \emptyset$. Hence, $\mathcal{L}(u) \downarrow \alpha = \{\epsilon\}$, so $(\psi(u))(\alpha) = \top$, from which the result follows. If $(\psi(v))(\alpha) = \rightarrow$, then there are two cases. If $\mathcal{L}(u) \downarrow \alpha \neq \{\epsilon\}$, then $(\psi(u))(\alpha) = \rightarrow$, from which the result follows. If $\mathcal{L}(u) \downarrow \alpha = \{\epsilon\}$, then it follows from Lemma 13.4 that $\mathsf{Up}(u, \alpha) = \emptyset$. Hence, $(\psi(u))(\alpha) = \top$, from which the result follows.

To show that ψ is a solution of minimal size, we need to show that for any other solution $h : S \rightarrow T_\Sigma$ of G, $\mathcal{D}(\psi(s)) \subseteq \mathcal{D}(h(s))$ for any s. This follows directly from Lemma 8. $\square$

6 An Algorithm

We have shown that the type inference problem for PTB can be reduced to solving constraint graphs. Using the characterization of Theorem 17, we get a straightforward type inference algorithm.

Theorem 18 *One can decide in time $O(n^3)$ whether a constraint graph of size n has a finite solution.*

Proof. By Theorem 17, there exists a finite solution if and only if the domain of the constructed solution ψ 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 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 $O(n^3)$. □

A λ-term of size n yields a constraint graph with $O(n)$ nodes and $O(n)$ edges. Allowing types to be represented succinctly by the automata $\mathcal{M}_s$, we get our main result.

Corollary 19 *The type inference problem for* PTB *is solvable in* $O(n^3)$ *time.*

Recursive types are just regular trees [1]. The minimal-size solution we have constructed, although possibly infinite, is a regular tree for every node in the constraint graph. Thus, we get the following result.

Corollary 20 *For* PTB_μ, *we can infer a minimal-size type annotation in* $O(n^3)$ *time.*

It remains to be seen how to extend this result to $\mathsf{PTB}_\mu \cup \{\mathsf{Int}\}$.

7 Example

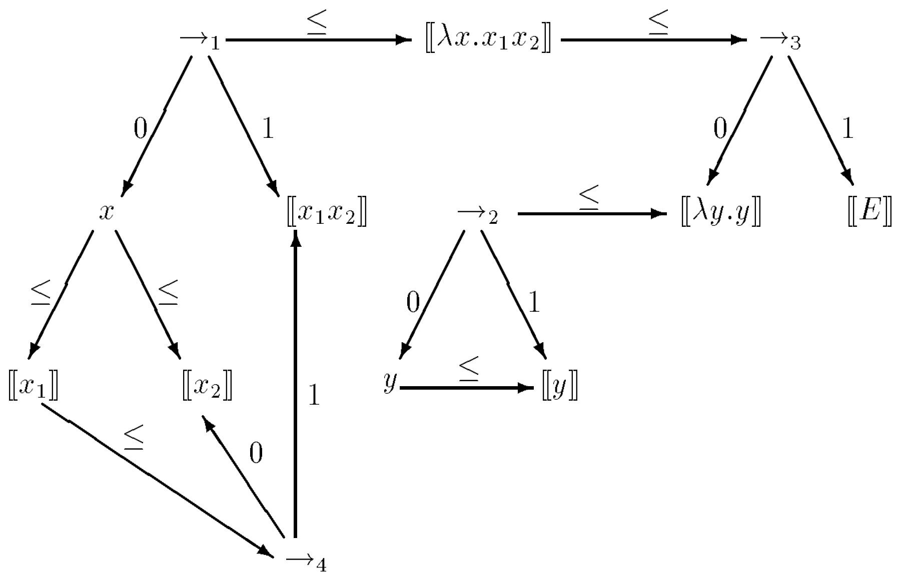
We now give an example of how our type inference algorithm works. Consider the λ-term $E = (\lambda x.xx)(\lambda y.y)$ which was also treated in [5]. We give each of the two occurrences of x a label so that the λ-term reads $(\lambda x.x_1x_2)(\lambda y.y)$. The constraint system $T(E)$ looks as follows:

From $\lambda x.x_1x_2$	$x \to [\![x_1x_2]\!] \leq [\![\lambda x.x_1x_2]\!]$
From $\lambda y.y$	$y \to [\![y]\!] \leq [\![\lambda y.y]\!]$
From E	$[\![\lambda x.x_1x_2]\!] \leq [\![\lambda y.y]\!] \to [\![E]\!]$
From x_1x_2	$[\![x_1]\!] \leq [\![x_2]\!] \to [\![x_1x_2]\!]$
From x_1	$x \leq [\![x_1]\!]$
From x_2	$x \leq [\![x_2]\!]$
From y	$y \leq [\![y]\!]$

To the left of the constraints, we have indicated from where they arise. We will use the following abbreviations:

The symbol:	abbreviates:
$\rightarrow_1$	$x \rightarrow [\![x_1 x_2]\!]$
$\rightarrow_2$	$y \rightarrow [\![y]\!]$
$\rightarrow_3$	$[\![\lambda y.y]\!] \rightarrow [\![E]\!]$
$\rightarrow_4$	$[\![x_2]\!] \rightarrow [\![x_1 x_2]\!]$

The constraint graph derived from $T(E)$ looks as follows:



To close the constraint graph, it is sufficient to first fill in the edges

$$\begin{aligned} [\![x_1 x_2]\!] &\stackrel{\leq}{\rightarrow} [\![E]\!] \\ [\![\lambda y.y]\!] &\stackrel{\leq}{\rightarrow} x \\ [\![y]\!] &\stackrel{\leq}{\rightarrow} [\![x_1 x_2]\!] \\ [\![x_2]\!] &\stackrel{\leq}{\rightarrow} y \ , \end{aligned}$$

and then make $\leq$ transitive.

The constraint graph has 13 nodes, so the automaton $\mathcal{M}$ has $13^2 = 169$ states. The following picture shows some of the states reachable from the state (x, x).

$$
\begin{tikzcd}
(x, x) \arrow[r, "\epsilon"] \arrow[d, "\epsilon"] & (x, \llbracket x_1 \rrbracket) \arrow[r, "\epsilon"] \arrow[d, "\epsilon"] & (x, \rightarrow_4) \arrow[d, "\epsilon"] & \\
(\llbracket \lambda y.y \rrbracket, x) \arrow[r, "\epsilon"] \arrow[d, "\epsilon"] & (\llbracket \lambda y.y \rrbracket, \llbracket x_1 \rrbracket) \arrow[r, "\epsilon"] \arrow[d, "\epsilon"] & (\llbracket \lambda y.y \rrbracket, \rightarrow_4) \arrow[d, "\epsilon"] & \\
(\rightarrow_2, x) \arrow[r, "\epsilon"] & (\rightarrow_2, \llbracket x_1 \rrbracket) \arrow[r, "\epsilon"] & (\rightarrow_2, \rightarrow_4) \arrow[r, "1"] \arrow[d, "0"] & (\llbracket y \rrbracket, \llbracket x_1 x_2 \rrbracket) \\
 & & (\llbracket x_2 \rrbracket, y) &
\end{tikzcd}
$$

It turns out that

$$
\begin{aligned}
\mathcal{L}(x) &= \{\epsilon, 0, 1\} \\
\mathcal{L}(y) &= \{\epsilon\} \\
\mathsf{Up}(x, 0) &= \emptyset \\
\mathsf{Up}(x, 1) &= \emptyset \\
\mathsf{Up}(y, \epsilon) &= \emptyset\ .
\end{aligned}
$$

We then get

$$
\begin{aligned}
\psi(x) &= \top \rightarrow \top \\
\psi(y) &= \top\ ,
\end{aligned}
$$

and it turns out that E is typable.

For comparison, we can apply the algorithm for type inference for PTB extended with recursive types [5] to E. The result is that both x and y get annotated by the infinite type $\mu\alpha.\alpha \to \alpha$ [5].

References

[1] Roberto M. Amadio and Luca Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, 1993. Also in Proc. POPL'91.

[2] Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient recursive subtyping. *Mathematical Structures in Computer Science*. To appear. Also in Proc. POPL'93, Twentieth Annual SIGPLAN–SIGACT Symposium on Principles of Programming Languages, pages 419–428, Charleston, South Carolina, January 1993.

[3] Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient inference of partial types. *Journal of Computer and System Sciences*, 49(2):306–324, 1994. Also in Proc. FOCS'92, 33rd IEEE Symposium on Foundations of Computer Science, pages 363–371, Pittsburgh, Pennsylvania, October 1992.

[4] John C. Mitchell. Type inference with simple subtypes. *Journal of Functional Programming*, 1:245–285, 1991.

[5] Jens Palsberg and Patrick M. O'Keefe. A type system equivalent to flow analysis. In *Proc. POPL'95, 22nd Annual SIGPLAN–SIGACT Symposium on Principles of Programming Languages*, pages 367–378, San Francisco, California, January 1995.

[6] Jens Palsberg and Michael I. Schwartzbach. Safety analysis versus type inference for partial types. *Information Processing Letters*, 43:175–180, 1992.

[7] Satish Thatte. Type inference with partial types. In *Proc. International Colloquium on Automata, Languages, and Programming 1988*, pages 615–629. Springer-Verlag (*LNCS* 317), 1988.

[8] Mitchell Wand. Type inference for record concatenation and multiple inheritance. *Information and Computation*, 93(1):1–15, 1991.

[9] Mitchell Wand, Patrick M. O'Keefe, and Jens Palsberg. Strong normalization with non-structural subtyping. *Mathematical Structures in Computer Science*. To appear.