Type Reconstruction with Recursive Types and Atomic Subtyping

Jerzy Tiuryn*
Institute of Informatics
Warsaw University
Banacha 2, 02-097 Warsaw
Poland
tiuryn@mimuw.edu.pl

Mitchell Wand†
College of Computer Science
Northeastern University
360 Huntington Avenue, 161CN
Boston, MA 02115, USA
wand@flora.ccs.northeastern.edu

Abstract

We consider the problem of type reconstruction for λ-terms over a type system with recursive types and atomic subsumptions. This problem reduces to the problem of solving a finite set of inequalities over infinite trees. We show how to solve such inequalities by reduction to an infinite but well-structured set of inequalities over the base types. This infinite set of inequalities is solved using Büchi automata. The resulting algorithm is in *DEXPTIME*. This also improves the previous *NEXPTIME* upper bound for type reconstruction for finite types with atomic subtyping. We show that the key steps in the algorithm are *PSPACE*-hard.

1 Introduction

John Mitchell, in his seminal paper [8, 9], considered a system for type reconstruction for λ-terms in which the set of types is augmented with a partial order (the subtype order), and the type inference rules are augmented with the *subsumption rule*

$$\frac{A \vdash M : s \quad s \leq t}{A \vdash M : t}$$

In this case the type reconstruction problem reduces to the problem of solving a set of inequalities over the set of types. Mitchell showed that if the partial order is generated by a set of *atomic coercions* on the base types, it reduces to the problem of solving a set of inequalities over the base types [9].

This paper has been the source of a considerable body of work [5, 7, 17, 14]. Such a system is an important component of a type-checking system for object-oriented programming. However, a good model of object-oriented programming must include *recursive* types, which correspond to infinite trees [2, 3], but Mitchell's algorithm applies only to well-founded types, which correspond to finite trees.

*This work was partly supported by NSF grants CCR-9002253 and CCR-9113196 and by Polish KBN grant No. 2 1192 91 01

†Work supported by the National Science Foundation under grants CCR-9002253 and CCR-9014603.

In this paper we show how to extend Mitchell's algorithm to handle recursive types. Instead of solving inequalities over finite trees, we will need to solve inequalities over possibly infinite trees. Instead of reducing tree inequalities to a finite set of "flat" inequalities over the base types, we will get an infinite but *regular* set of flat inequalities. Instead of solving these inequalities in the base order (as in [17] or [7]) by nondeterministic choice, we solve them by reducing to the emptiness problem for Büchi automata. The resulting algorithm is in *DEXPTIME*. By contrast, the best previously-known upper-bound for type reconstruction with atomic subtyping, in the case of well-founded types, is *NEXPTIME*; our algorithm can be used for this case also. Last, we show that the key steps in the algorithm are *PSPACE*-hard.

Definitions are given in Section 3, along with the basic properties of the order on infinite trees. The decision problems are posed in Section 4. Then, in Section 5, we begin the development of the algorithm. The algorithm has four main steps:

1. Reduce the type reconstruction problem to a set of inequalities over finite trees. This is the same as for the finite case. We sketch this familiar reduction in Section 5.
2. Find the shapes of the solutions via unification. The algorithm is presented in Section 6.1.
3. Enumerate the frontiers of the shapes to generate an infinite but *regular* set of flat inequalities. This step is presented in Section 6.2.
4. Solve inequalities over the partial order on the base types. This is done in Section 7 by reduction to Büchi automata, whose emptiness problem is solvable in polynomial time [15].

We note that our definition of types includes non-regular as well as regular trees; we obtain as a corollary that if an expression has any typing at all then it has one in which all the types are regular.

The resulting algorithm is in *DEXPTIME*, as all the steps are polynomial except for the reduction to Büchi automata, which is $2^{O(n)}$. On the other hand, when C is discrete, then *C-TR* reduces to unification on infinite trees and is therefore in *PTIME*.

We then present some lower bounds in Section 8. We show that *C-REG-SAT* is *PSPACE*-hard for every non-trivial poset C by reduction from quantified boolean formulas to the termination problem for a class of automata called autonomous reading pushdown automata (ARPDA), and then from ARPDA termination to *C-REG-SAT*.

2 Related Work

Mitchell [8, 9] introduced the problem of type reconstruction with coercions, including atomic coercions, and sketched the main algorithms for the case of well-founded types. This work concentrated on generating the set of atomic coercions that must hold among the base types. Fuh and Mishra [5] expanded these algorithms and introduced the variant in which the set of atomic coercions was either fixed or was part of the input.

Wand and O'Keefe [17] showed that type reconstruction when the set of atomic coercions was part of the input was NP-hard if certain constants were allowed in the

terms to be typed. Mitchell and Lincoln [7] improved this result by establishing NP-hardness without constants, and by systematically considering the various versions of the problem.

Tiuryn [14] considered the problem of satisfiability of subtype inequalities (what we call *C-TREE-SAT*, but over finite trees only) and showed that for some classes of posets, the problem is *PSPACE*-hard, but for others it is polynomial-time.

All this work concerned well-founded (finite) types only. Amadio and Cardelli [1] considered a related problem for infinite types. They considered the validity problem for expressions denoting regular types, but with a rather different order, in which there were elements $\bot$ and $\top$ which were bounds for all types. This order may be related to the "partial types" of Thatte [13], which have a top (but not a bottom element). The decidability of type reconstruction for this type discipline was shown for the well-founded case by O'Keefe and Wand [10]. Kozen, Palsberg, and Schwartzbach [6] gave an $O(n^3)$ algorithm both for finite types and for recursive types under the partial-type ordering.

3 Definitions

3.1 Trees

Given a set C of labels, the set $Trees_C$ is the set of binary trees with leaf labels chosen from C; that is, the set of non-empty partial functions $t : \{0,1\}^* \rightharpoonup (C \cup \{\rightarrow\})$ such that

1. the domain of t is prefix-closed,
2. if $t(\alpha) =$ "$\rightarrow$" then $t(\alpha 0)$ and $t(\alpha 1)$ are both defined, and
3. if $t(\alpha) \in C$ then neither $t(\alpha 0)$ nor $t(\alpha 1)$ is defined.

Given a tree, its *shape* is its domain, that is, the set of nodes or paths in the tree. We will occasionally refer to a string in $dom(t)$ as a "path" or an "address". We say π is a *leaf* of t if it is in $dom(t)$, but neither $\pi 0$ nor $\pi 1$ is in $dom(t)$.

We will write $t{\downarrow}w$ for the subtree of t rooted at address w, that is the tree defined by $dom(t{\downarrow}w) = \{\pi \mid w\pi \in dom(t)\}$ and $(t{\downarrow}w)(\pi) = t(w\pi)$. The set $Regtrees_C$ of *regular* trees is the set of trees with only finitely many distinct subtrees $t{\downarrow}w$. Such trees can be thought of as being generated by a finite automaton.

3.2 Partial Order on Trees

We assume we are given a partial order $\leq_C$ on the label set C. This relation is extended to trees as follows:

1. $t \leq_0 t'$ for all t, t'.
2. For each $n \geq 0$, $\leq_{n+1}$ is defined as follows:

$$\frac{c \leq_C c'}{c \leq_{n+1} c'}$$

$$\frac{s' \leq_n s \quad t \leq_n t'}{(s \rightarrow t) \leq_{n+1} (s' \rightarrow t')}$$

3. $s \leq t$ iff $s \leq_n t$ for all $n \geq 0$.

This definition replaces the usual "bottom up" definition for $\leq$ on finite trees by a "top-down" definition. The subscripts essentially require that $s \leq_n t$ iff $s \leq t$ down to n levels; by quantifying over n, we require that $s \leq t$ for all levels. This intuition is made precise by the following lemmas.

The same-shape property, familiar from finite trees, extends to infinite trees as well:

Lemma 1 *If $t \leq t'$, then $dom(t) = dom(t')$.*

Proof: This is done by induction on the length of addresses, using the following lemma: for all $n \geq 0$, if $t \leq_n t'$ and $|\pi| \leq n$, then $\pi \in dom(t)$ iff $\pi \in dom(t')$. This is an easy induction on n. The base case uses the fact that tree domains always contain ϵ. □

Let *POS* denote the regular set of strings in $\{0,1\}^*$ with an even number of 0's, and let *NEG* denote the corresponding set with an odd number of 0's. The following easy lemma will also be useful:

Lemma 2 *Let $t \leq t'$ and $\pi \in POS$ (resp. NEG). If $\pi \in dom(t)$ then $t{\downarrow}\pi \leq t'{\downarrow}\pi$ (resp. $t'{\downarrow}\pi \leq t{\downarrow}\pi$).*

Lemma 3 *$t \leq t'$ iff $dom(t) = dom(t')$ and for every leaf π of t either*

1. *$\pi \in POS$ and $t(\pi) \leq_C t'(\pi)$*
2. *$\pi \in NEG$ and $t'(\pi) \leq_C t(\pi)$*

4 The decision problems

4.1 *C-TR*

Let the set of types be $Trees_{C \cup X}$ for some set C of base types and some set X of type variables.

The problem *C-TR* has as input a triple (A, M, t), where A is a map from a finite set of variables of the λ-calculus to regular types (represented as non-deterministic finite automata), M is a λ-term, and t is a type. The problem is to determine whether there exists a map B and a substitution $\sigma : X \to Trees_C$ such that $B \supseteq A$ and $B\sigma \vdash M : t\sigma$ is deducible in the following system:

$$A \vdash x : A(x)$$

$$\frac{A \vdash M : t \to t' \quad A \vdash N : t}{A \vdash (M\,N) : t'}$$

$$\frac{A[x : t] \vdash M : t'}{A \vdash (\lambda x.M) : t \to t'}$$

$$\frac{A \vdash M : t \quad t \leq t'}{A \vdash M : t'}$$

Here t and t' range over $Trees_C$, and A and B range over maps from a finite set of variables of the λ-calculus to regular types (represented as non-deterministic finite automata).

This version of the problem does not include constants in the λ-terms. The problem including constants can be reduced to *C-TR* by including the types of the constants in A. When C is a discrete order ($c \leq_C c'$ implies $c = c'$), this is the ordinary type reconstruction problem over infinite trees. Another variant of this problem has as input only A and M, and asks whether t exists; this problem reduces to *C-TREE-SAT* similarly.

All these questions can be asked when the types are finite trees (i.e. simple types) only; we denote the finite-tree version of *C-TR* by $C\text{-}TR_F$.

4.2 *C-TREE-SAT*

Given a partial order C on the constants, the problem *C-TREE-SAT* is: Given a finite set of inequalities of the form $t \leq t'$ where t and t' range over terms of the form

$$t ::= c \mid x \mid t \to t'$$

is there a valuation $\sigma : Vars \to Trees_C$ that satisfies all the inequalities? When C is discrete, this is just unification on infinite trees, and it is well-known that it is decidable in polynomial time, and if a solution exists, then there is a solution in which all the trees are regular.

We will use Σ as a symbol to range over instances of *C-TREE-SAT* and similar problems. We use x, y, z as metavariables ranging over the variables in the inequalities. For *C-REG-SAT* below, we will introduce x_α as subscripted variables, and we will identify x and x_ϵ.

Let us consider an example, which we will use throughout to illustrate the pieces of the algorithm. Consider the inequality

$$x \leq y \to (c \to x)$$

By repeatedly applying Lemma 2, it is easy to deduce that in any solution σ, we will have

$$\begin{aligned} &\sigma y \leq (\sigma x)\!\downarrow\! 0 \\ &c \leq (\sigma x)\!\downarrow\! 10 \\ &(\sigma x)\!\downarrow\! 0 \leq (\sigma x)\!\downarrow\! 110 \\ &(\sigma x)\!\downarrow\! 10 \leq (\sigma x)\!\downarrow\! 1110 \end{aligned}$$

etc. Furthermore, since x and x_1 are known to be interior nodes, all of these addresses must be in the domain of any solution, and all of $(\sigma x)\!\downarrow\! 11^*10$ must be leaves comparable to c, forming an increasing chain. In general, we have

$$\{(\sigma x)\!\downarrow\!\alpha 0 \leq (\sigma x)\!\downarrow\!\alpha 110 \mid \alpha \in 1^*\}$$

By more complex initial conditions, one can generate quite complex sets of constraints, with many interlocking chains of inequalities. Our goal is to reduce *C-TREE-SAT* to an

infinite (but structured) set of constraints to be solved in the partial order C. This leads us to *C-REG-SAT*.

4.3 *C-REG-SAT*

Definition 1 *A set of constraints is* regular *iff it can be expressed as a finite union of sets of inequalities of the following forms:*

(1) $\{x_{w\pi} \leq y_{w'\pi} \mid \pi \in R\}$ *for some regular set* R.

(2) $\{x_w \leq c\}$ *for some constant* c

(3) $\{c \leq x_w\}$ *for some constant* c.

Note that a regular set of constraints is a "flat" system: it contains no arrows, so we may consider solving it over C, not $Trees_C$.

The problem *C-REG-SAT* is: Given a regular set of constraints, with the regular sets R represented by nondeterministic finite automata, is there a valuation $\sigma : Vars \to C$ that satisfies all the inequalities?

We will show the decidability of *C-REG-SAT* by reducing it to the emptiness problem for Büchi automata.

The fragment of *C-REG-SAT* in which all the regular sets R are finite is denoted *C-FIN-SAT*.

5 Reducing *C-TR* to *C-TREE-SAT*

The reduction from ordinary type reconstruction to unification on finite trees is well-known (e.g. [16]). The same process can be used to reduce *C-TR* to *C-TREE-SAT*.

Given an instance (A, M, t) of *C-TREE-SAT*, assign a type variable to every subexpression of M and every binding occurrence of a variable in M. We write t_N for the type variable associated with subexpression N; technically we should distinguish different occurrences of N, but this will be clear from context.

Since $\leq$ is a partial order, consecutive occurrences of the subsumption rule may be merged. Therefore, if σ is any solution to (A, M, t), then $A\sigma \vdash M : t\sigma$ has a derivation tree in which each "structural" step is followed by exactly one subsumption step. For example, for an application, the tree would look like:

$$\dfrac{\dfrac{A\sigma \vdash M : t_M\sigma \quad A\sigma \vdash N : t_N\sigma}{A\sigma \vdash (M\ N) : t} \quad t \leq t_{(MN)}\sigma}{A\sigma \vdash (M\ N) : t_{(MN)}\sigma}$$

where t is some type. We can summarize this information by generating the inequalities

$$t_M = t_N \to t$$
$$t \leq t_{(MN)}$$

where t is a fresh type variable.

Extending these considerations to the other cases gives the following set of rules:

For each	generate
x	$t_{A(x)} \leq t_x$
$\lambda x.M$	$t_x \rightarrow t_M \leq t_{\lambda x.M}$
(MN)	$t_M = t_N \rightarrow t_1$
	$t_1 \leq t_{(MN)}$

where $t_A(x)$ is the type variable associated with the binding occurrence of x and t_1 is a fresh type variable.

Each solution to the generated set of inequalities corresponds to a type inference tree, and vice versa. Hence *C-TR* reduces to *C-TREE-SAT*.

6 Reducing *C-TREE-SAT* to *C-REG-SAT*

6.1 Finding the shape of the solution

By Lemma 1, we can determine the shapes of any solution to *C-TREE-SAT* by reducing to the familiar problem of unification over infinite trees. More precisely, given an instance Σ of *C-TREE-SAT*, we can produce an instance *Shape*(Σ) of unification over infinite trees as follows:

1. Replace every constant appearing in Σ by a single constant c_0. For each term t, call the resulting term $\bar{t}$
2. Replace every inequality $t \leq t'$ in Σ by the equality $\bar{t} = \bar{t}'$.

Lemma 4 *If σ is any solution to Σ, then the map σ' defined by*

$$(\sigma' x)(\pi) = \begin{cases} c_0 & \text{if } (\sigma x)(\pi) \in C \\ (\sigma x)(\pi) & \text{otherwise} \end{cases}$$

is a solution to Shape(Σ).

Proof: Obvious from Lemma 1. □

We say Σ is *shape-consistent* iff *Shape*(Σ) is solvable.

Lemma 5 *If Σ is not shape-consistent, then Σ is unsatisfiable.*

Proof: Immediate from Lemma 4. □

By the familiar algorithm ([4], Theorem 4.9.2), we can determine if *Shape*(Σ) is solvable and, if it is, we can construct a principal solution to *Shape*(Σ), that is a map σ_Σ : *Vars* $\rightarrow$ *Regtrees*$_{C \cup X}$ for some finite set X of new variables, such that the solutions to *Shape*(Σ) are precisely the maps of the form $\sigma \circ \tau$, where τ is any map $X \rightarrow$ *Trees*$_C$.

Therefore, for each variable x appearing in Σ, we can construct regular sets $L_\Sigma(x)$, $Int_\Sigma(x)$ and $C_\Sigma(x)$ with the following properties.

$$\begin{aligned}
\pi \in L_\Sigma(x) &\iff \pi \in dom(\sigma_\Sigma x) \\
&\iff \text{for every solution } \sigma \text{ of } Shape(\Sigma),\ \pi \in dom(\sigma x) \\
\pi \in Int_\Sigma(x) &\iff (\sigma_\Sigma x)(\pi) = \rightarrow \\
&\iff \text{for every solution } \sigma \text{ of } Shape(\Sigma),\ (\sigma x)(\pi) = \rightarrow \\
\pi \in C_\Sigma(x) &\iff (\sigma_\Sigma x)(\pi) = c_0 \\
&\iff \text{for every solution } \sigma \text{ of } Shape(\Sigma),\ (\sigma x)(\pi) = c_0
\end{aligned}$$

Let us further define $Leaves_\Sigma(x) = L_\Sigma(x) - Int_\Sigma(x)$. Furthermore, any solution σ, and the functions L_Σ, Int_Σ, etc., can be extended to act on finite terms instead of just on variables by setting $L_\Sigma(s \rightarrow t) = L_\Sigma(s) \rightarrow L_\Sigma(t)$, etc. Then, if $(s = t) \in \Sigma$, we have $L_\Sigma(s) = L_\Sigma(t)$, etc.

For our example, we have $L_\Sigma(x) = 1^* \cup 1^*0$, $Int_\Sigma(x) = 1^*$, $Leaves_\Sigma(x) = 1^*0$, and $C_\Sigma(x) = (11)^*10$.

Lemma 6 *Let Σ be a shape-consistent instance of C-*TREE-SAT. *Then:*

1. *If $\pi \in L_\Sigma(x)$, then in any solution σ of Σ, $\pi \in dom(\sigma(x))$.*
2. *If $\pi \in Int_\Sigma(x)$, then in any solution σ of Σ, $(\sigma x)(\pi) = \rightarrow$.*
3. *If $\pi \in C_\Sigma(x)$, then in any solution σ of Σ, $(\sigma x)(\pi)$ is a constant.*

Proof: We will do part 3; the others are similar. Let $\pi \in C_\Sigma(x)$ and σ be any solution of Σ. Form σ' as in Lemma 4. Since σ' is a solution to $Shape(\Sigma)$, we know that $(\sigma' x)(\pi) = c_0$. But this implies that $(\sigma x)(\pi) = c$ for some $c \in C$, by the construction of σ'. □

6.2 Enumerating the leaf inequalities

Now we can give the reduction from *C-TREE-SAT* to *C-REG-SAT*. For a shape-consistent instance Σ of *C-TREE-SAT*, we build an instance $Flat(\Sigma)$ of *C-REG-SAT* by the following process. We start with the set Σ of inequalities with variables x, y, etc., and build a new set of inequalities $\hat{\Sigma}$ over subscripted variables x_w for $w \in \{0,1\}^*$; we identify x and x_ϵ.

1. For each inequality $(s \leq t) \in \Sigma$, consider each pair of strings (w, w') such that w is a leaf of s and ww' is a leaf of t.
2. Consider the case in which $s(w)$ is a variable (say x), and ww' is a leaf (either $t(ww') = c$ or $t(ww') = y$). If ww' is positive, insert in $\hat{\Sigma}$ the inequality $x_{w'} \leq c$ or $x_{w'} \leq y$. If ww' is negative, insert in $\hat{\Sigma}$ the inequality $c \leq x_{w'}$ or $y \leq x_{w'}$.
3. If $s(w)$ is a constant c, it must be that $w' = \epsilon$ (otherwise Σ would not be shape-consistent) so $t(w) = c'$ or $t(w) = y$. If w is positive, insert in $\hat{\Sigma}$ the inequality $c \leq c'$ or $c \leq y$. If w is negative, insert in $\hat{\Sigma}$ that inequality $c' \leq c$ or $y \leq c$.

4. Similarly for each pair of strings (w, w') where w is a leaf of t and ww' is a leaf of s.

This gives us a set of inequalities of the form $x_w \leq y$, $x \leq y_w$, $x_w \leq c$, $c \leq x_w$, and $c \leq c'$.

For our example, this process generates $\hat{\Sigma} = \{y \leq x_0, c \leq x_{10}, x_{11} \leq x\}$.

Lemma 7 *If Σ is shape-consistent, then Σ is satisfiable iff $\hat{\Sigma}$ is satisfiable.*

Proof: If Σ has a solution σ, define $\hat{\sigma}(x_w) = (\sigma x)\downarrow w$. If $\hat{\sigma}$ is a solution to $\hat{\Sigma}$, define σx to be the smallest tree such that $(\sigma x)(ww') = (\hat{s}x_w)(w')$, by marking every prefix of w with $\rightarrow$. □

The instance *Flat*(Σ) of *C-REG-SAT* is defined as follows:

- For each inequality of the form $x_w \leq y$, include the regular constraints

$$\{x_{w\pi} \leq y_\pi \mid \pi \in C(y) \cap POS\}$$

and

$$\{y_\pi \leq x_{w\pi} \mid \pi \in C(y) \cap NEG\}$$

- Include each inequality of the form $x_w \leq c$ or $c \leq x_w$.

For our example $C_\Sigma(x) = (11)^*10 \subseteq NEG$, and $C_\Sigma(y) = \emptyset$, so we get *Flat*(Σ) = $\{c \leq x_{10}, \{x_\pi \leq x_{11\pi} \mid \pi \in (11)^*10\}\}$

Theorem 1 *If Σ is shape-consistent, then Σ is satisfiable iff Flat(Σ) is satisfiable.*

Proof: ($\Rightarrow$): If σ satisfies Σ and $\pi \in C_\Sigma(x)$, then $(\sigma x)(\pi)$ is a constant. Hence the variables in *Flat*(Σ) are all assigned values in C, and it is easy to see that all of the constraints in *Flat*(Σ) are satisfied.

($\Leftarrow$): Given a solution σ to *Flat*(Σ), construct a solution σ' to Σ as follows:

1. For each variable x in Σ, let $dom(\sigma' x) = dom(\sigma_\Sigma x) = L_\Sigma(x)$.
2. If $\pi \in Int_\Sigma(x)$, let $(\sigma' x)(\pi) = \rightarrow$.
3. If $\pi \in C_\Sigma(x)$, let $(\sigma' x)(\pi) = \sigma(x_\pi)$ We will prove that x_π is a variable in *Flat*(Σ).
4. Choose a $c_0 \in C$. If $\pi \in Leaves(x) - C(x)$, let $(\sigma' x)(\pi) = c_0$.

Since Σ is shape-consistent, it follows that $Int_\Sigma(x) \cap C_\Sigma(x)$ is empty, so it is easy to see that this assigns a label to every address $\pi \in L_\Sigma(x)$.

We must show that σ is a solution to Σ. Let $(s \leq t) \in \Sigma$. Then $(\bar{s} = \bar{t}) \in Shape(\Sigma)$, so by the construction of σ', $dom(\sigma' s) = dom(\sigma_\Sigma \bar{s}) = dom(\sigma_\Sigma \bar{t}) = dom(\sigma' t)$. By Lemma 3, it is enough to show that for every leaf π of $dom(\sigma' s)$, $(\sigma' s)(\pi)$ and $(\sigma' t)(\pi)$ are appropriately related.

If $\pi \in Leaves_\Sigma(s) - C_\Sigma(s)$, then $(\sigma' s)(\pi) = (\sigma' t)(\pi) = c_0$, so the condition of Lemma 3 is satisfied regardless of whether π is positive or negative.

The remaining case is that $\pi \in C_\Sigma(s) = C_\Sigma(t)$. Then there must be paths w_1, w_2, π_1, π_2 such that $\pi = w_1\pi_1$ and w_1 is a leaf of s, and $\pi = w_2\pi_2$ and w_2 is a leaf of t.

If $s(w_1)$ is a constant (say c), then $\pi_1 = \epsilon$. So $t(w_2)$ must either be some constant c', in which case $\pi_2 = \epsilon$, or some variable y. Consider the case in which $\pi = w_1$ is positive. Then $Flat(\Sigma)$ includes $c \leq c'$ or $c \leq y_{\pi_2}$. Since σ is solution to $Flat(\Sigma)$, we have $c \leq_C c'$ or $c \leq (\sigma y_{\pi_2})$. In either case we have $(\sigma' s)(\pi) \leq (\sigma' t)(\pi)$ as required. The case for π negative is symmetrical.

So assume that $s(w_1)$ is some variable x, and $t(w_2)$ is some variable y. Then we have $(\sigma' s)(\pi) = (\sigma' x)(\pi_1)$ and $(\sigma' t)(\pi) = (\sigma' y)(\pi_2)$.

Without loss of generality, assume that w_1 is a prefix of w_2, say $w_2 = w_1 w$. Then we have $w_1\pi_1 = \pi = w_2\pi_2 = w_1 w\pi_2$, so $\pi_1 = w\pi_2$.

We now have four cases, depending on the parity of w and π_2. We will do only the case where both are positive. Since w is positive, $\hat{\Sigma}$ must contain the inequality $x_w \leq y$. Now $\pi \in C_\Sigma(s)$, so $\pi_2 \in C_\Sigma(y)$, $w\pi_2 \in C_\Sigma(x)$, and $(x_{w\pi_2} \leq y_{\pi_2}) \in Flat(\Sigma)$. Therefore σ assigns a value from C to each of these variables, as desired. Furthermore, we observe $(\sigma' s)(\pi) = (\sigma' x)(\pi_1) = (\sigma' x)(w\pi_2) = (\sigma x_{w\pi_2}) \leq (\sigma y_{\pi_2}) = (\sigma' y)(\pi_2) = (\sigma' t)(\pi)$, establishing the necessary relation between $(\sigma' s)(\pi)$ and $(\sigma' t)(\pi)$. The other cases are similar, reversing the signs as needed. □

7 Reducing *C-REG-SAT* to Büchi automata

A Büchi automaton is a nondeterministic automaton which walks down a possibly infinite tree in which every node has a label chosen from some alphabet A. A run associates each node with a state. The state at any node may depend non-deterministically on the state of the machine at the parent node, the label at the parent node, and the direction (0 or 1) taken from the parent node to the current node.

Formally, the automaton is specified by a tuple (Q, q_0, Δ, F), consisting of a finite set of Q states, an initial state $q_0 \in Q$, a transition relation $\Delta \subset Q \times A \times \{0,1\} \times Q$ and a set $F \subset Q$ of final states. a run on a tree t is a labelled tree t' with the same domain as t, such that $t'(\epsilon) = q_0$ and for any address π in the interior of t, the tuple $(t'(\pi), t(\pi), a, t'(\pi\alpha))$ is in the set for $a \in \{0,1\}$. The run is successful if on each path, some final state occurs infinitely often. It is well-known that the emptiness problem for Büchi automata is decidable, and is in fact decidable in polynomial time [12, 15].

Given a regular set of inequalities over C, we will construct a Büchi automaton whose language is non-empty iff the set of inequalities is satisfiable. Our machines will in fact be deterministic.

The first step is to reverse all the indices in Σ. This gets us to a finite set of families of inequalities of the form

$$\{x_{\pi w} \leq y_{\pi w'} \mid \pi \in R\}$$

for some regular set R represented as a nondeterministic finite automaton. This transformation clearly preserves satisfiability. We call such a set of inequalities *reverse-regular*.

Theorem 2 *Given any reverse-regular set of inequalities Σ, one can construct a Büchi automaton $\mathcal{A}$ such that the set of trees accepted by $\mathcal{A}$ is non-empty iff Σ is satisfiable.*

Proof: Without loss of generality, we consider only families of the form

$$\{x_{\pi w} \leq y_\pi \mid \pi \in R\}$$

and

$$\{x_\pi \leq y_{\pi w'} \mid \pi \in R\}$$

The constraints constructed in the preceding reduction are of this form; in general, any set of the form $\{x_{\pi w} \leq y_{\pi w'} \mid \pi \in R\}$ can be replaced by $\{x_{\pi w} \leq z_\pi \mid \pi \in R\}$, and $\{z_\pi \leq y_{\pi w'} \mid \pi \in R\}$, for some new variable z.

Assume that there are n unsubscripted variables $x^1, x^2, \ldots x^n$ in Σ, that is, the variables in Σ are of the form x^i_w for some $i \in \{1 \ldots n\}$. We will run our automaton over complete binary trees labelled by elements of C^n.

Such a tree will correspond to a solution of the set of inequalities. These trees are not quite solutions to the original set of inequalities over trees, because the indices have been reversed.

Each family of inequalities

$$\{x^i_\pi \leq x^j_{\pi w} \mid \pi \in R\}$$

can be represented by the tuple $(i, j, w, +, R)$. Similarly, each family of inequalities $\{x^i_{\pi w} \leq x^j_\pi \mid \pi \in R\}$ can be represented by the tuple $(j, i, w, -, R)$. In each case the first element of the tuple indicates the variable with the shorter subscript. The sign indicates whether the "later-found" element is larger or smaller than the "earlier-found" one. We refer to these as the *original items.*

Each inequality $c \leq x^i_w$ can be represented by the tuple $(c, i, w, +)$, and each inequality $x^i_w \leq c$ can be represented by the tuple $(c, i, w, -)$ We refer to these tuples collectively as *items.*

We construct an automaton $\mathcal{A}$ whose states are either (a) a distinguished failure state or (b) a finite set of items. The initial state will be the set of items corresponding to the constraints of the form $c \leq x^i_w$ and $x^i_w \leq c$. The accepting states will be all sets other than the failure state.

Once in the failure state, the machine will stay in the failure state forever. Otherwise, at every node the machine splits into two states, one for each branch. We refer to these states as the 0-successor state and the 1-successor state, respectively.

To construct the set of items for the two successor states, add items according to the following rules, beginning with the empty set:

1. For each original item $(i, j, aw, +, R)$, if the current address is in R then put the item $(c^i, j, w, +)$ in the a-successor state. Similarly for each item of the form $(i, j, aw, -, R)$.

2. For each original item $(i, j, \epsilon, +, R)$, if the current address is in R then check to see if $c^i \leq c^j$. If not, then make each a-successor state ($a = 0$ or 1) the failure state. (If $c^i \leq c^j$, then this constraint is satisfied at this address, so no item need be inserted.) Similarly for the original item $(i, j, \epsilon, -, R)$.

3. For each item $(c, i, aw, +)$ in the state, then the item $(c, i, w, +)$ will be in the a-successor state ($a = 0$ or 1).

4. For each item $(c, i, aw, -)$ in the state, then the item $(c, i, w, -)$ will be in the a-successor state.

5. For each item $(c, i, \epsilon, +)$ in the state, let c^i be the i-th component of the label at the current node. If $c^i \not\leq c$, then make both successor states the failure state. (If $c^i \leq c$, then the constraint coded by this item has been satisfied, so the item can be deleted). Similarly for each item $(c, i, \epsilon, -)$.

For each family of inequalities, represented by $(i, j, w, \pm, R)$, the automaton $\mathcal{A}$ keeps track of the current address in the tree and check to see whether it is in the regular set R. If the current address is in R, then we create an item $(c, j, w, \pm)$ that will walk down along the path w and check to see if the j-th component at that location satisfies the necessary inequality.

Each item $(c, j, w, \pm)$ walks down the tree from its creation point π, following path w to the tree address πw. It then compares the value of the y component at πw to c and either succeeds or fails, depending on the value of the $\pm$. If the constraint is violated, then the machine enters a failure state and rejects the input. Otherwise, the machine continues.

We next count the number of possible items $(c, j, w, \pm)$. Let the system Σ have k groups of inequalities, each of the form $\{c \leq x_{w_i}\}$, $\{x_{w_i} \leq c\}$, $\{x_\pi \leq y_{\pi w_i} \mid \pi \in R\}$, or $\{y_{\pi w_i} \leq x_\pi \mid \pi \in R\}$. Then the number of possible items obtained from one such group is at most $|C| \cdot |w_i|$. So the total number of items is at most

$$|C| \cdot (|w_1| + \ldots + |w_k|) \leq |C| \cdot |\Sigma|$$

Thus the number of possible items is $O(|\Sigma|)$, so $\mathcal{A}$ has at most $2^{O(|\Sigma|)}$ states.

We next show that this machine accepts some tree iff the reverse-regular set Σ of inequalities is satisfiable. If $\sigma : \mathit{Vars}(\Sigma) \to C$ is a solution to Σ, construct a tree $t : \{0,1\} \to C^n$ by setting $t(\pi) = (c_1, \ldots c_n)$, where $c_i = \sigma(x_\pi^i)$ if $x_\pi^i \in \mathit{Vars}(\Sigma)$ and $c_i = c_0$ (some fixed constant) otherwise. This tree will be accepted by $\mathcal{A}$, since it will never send $\mathcal{A}$ to the failure state.

Conversely, if $t : \{0,1\} \to C^n$ is accepted by $\mathcal{A}$, then for each $x_\pi^i \in \mathit{Vars}\,\Sigma$ let $\sigma(x_\pi^i)$ be the i-th component of $t(\pi)$. Since no run of $\mathcal{A}$ on t enters the failure state, it follows that all the partial-satisfaction conditions together with the component inequalities are satisfied, that is, this is a solution of Σ. □

Theorem 3 *1. Given a reverse-regular set of inequalities Σ, it is decidable in deterministic exponential time whether Σ is satisfiable.*

*2. C-*REG-SAT *is decidable in* DEXPTIME.

Proof: By the polynomial decidability of the emptiness problem for Büchi automata and the observation that the size of $\mathcal{A}$ is $2^{O(|\Sigma|)}$. □

Note by contrast that *C-FIN-SAT* is in NP, for every C.

We can summarize the sequence of reductions as follows:

Theorem 4 *1. The problem C-*TREE-SAT *is decidable in deterministic exponential time.*

*2. The problem C-*TR *is decidable in deterministic exponential time.*

*3. If Σ is an instance of C-*TR *that has a solution, then it has a solution in which all the types are regular trees.*

Proof: (i) Use nondeterministic finite automata to represent the regular sets in the solution of *Shape*(Σ); then all the reductions except the last are polynomial.

(ii) All the reductions except the last are polynomial.

(iii) Because if the language accepted by a Büchi automaton is nonempty, then it includes some regular tree. □

Theorem 5 *The problem C-*TR$_F$ *is decidable in deterministic exponential time.*

Proof: To use this algorithm for type reconstruction with atomic subtyping in the case of well-founded types, merely test each set $L_\Sigma(x)$ for finiteness. This can be done in polynomial time. □

This result improves the upper bound for *C-TR$_F$* from *NEXPTIME* to *DEXTIME*.

8 Lower Bounds

We show that if C is any nontrivial partial order (ie it has two unequal but comparable elements), then *C-REG-SAT* is *PSPACE*-hard. We will do this by defining a class of automata called *autonomous reading PDA*'s (ARPDA's). Then we show that the ARPDA termination problem is PSPACE-hard, and that ARPDA termination reduces to *C-REG-SAT* over any nontrivial partial order.

An ARPDA consists of a finite set Q of states and a pushdown stack over the alphabet $\{0,1\}$, so an instantaneous description of a machine state is a pair (q, w) with $q \in Q$ and $w \in \{0,1\}^*$; we depict the top of the stack as being at the right-hand end of w. The machine has an initial state q^0 and a final state q^f, and its behavior is specified by a set Δ of *transitions*. Each transition is of one of two forms:

1. A *pds transition* $((p, a) \mapsto (q, b))$, where $p, q \in Q$ and $a, b \in \{0, 1, \epsilon\}$.
2. A *pds query* $((p, R) \mapsto q)$, where $p, q \in Q$, and $R \subseteq \{0,1\}^*$ is a regular set, represented as a nondeterministic finite automaton.

An ARPDA is a nondeterministic machine. Its behavior relation $\rightarrow$ is defined as follows:

- If $((p, a) \mapsto (q, b)) \in \Delta$, then $(p, ua) \rightarrow (q, ub)$ for any $u \in \{0,1\}^*$.
- If $((p, R) \mapsto q) \in \Delta$, then $(p, u) \rightarrow (q, u)$ whenever $u \in R$.

The ARPDA termination problem is: Given an ARPDA M, does $(q^0, \epsilon) \stackrel{*}{\rightarrow} (q^f, \epsilon)$?

Theorem 6 *The ARPDA termination problem is PSPACE-hard.*

Proof: By reduction from evaluation of quantified boolean formulae. Let $(Q_1x_1)\ldots(Q_nx_n)\Phi$ be a quantified boolean formula; that is, each Q_i is a quantifier ($\forall$ or $\exists$) and Φ is a boolean formula in disjunctive normal form over the variables $\{x_1,\ldots x_n\}$. We construct a ARPDA M that terminates iff this formula is true. The machine works by traversing a backtracking search tree over $\{x_1,\ldots x_n\}$. It maintains its position in the tree by keeping the values of $x_1,\ldots,x_n$ on the stack. It keeps track of its direction of travel (down or up) and its current level in $\{0,\ldots n+1\}$ in its control state Q. The initial state is (down, 1), and the final state is (up, 0).

The machine maintains the invariant that in state (up, i), the stack contains a valuation $x_1,\ldots,x_i$ that makes the formula $(Q_{i+1}x_{i+1})\ldots(Q_nx_n)\Phi$ true.

We next describe what happens at each state (d,i), when the machine is at level i travelling in direction d, and at the same time show that the machine maintains this invariant.

(down, i) On this visit, the machine is searching down in the tree. If $i \leq n$ and Q_i is $\forall$, push a 0 on the stack. If $i \leq n$ and Q_i is $\exists$, nondeterministically choose a value for x_i and push it on the stack. In either case go to state (down, $i+1$).

If $i > n$, we have a complete set of values for $\{x_1,\ldots x_n\}$ on the stack. Evaluate the formula Φ using these values; this is possible by encoding Φ as a nondeterministic finite automaton and using the ability of M to check whether its stack matches an arbitrary regular set. If the formula is true, go to state (up, n). If not, then loop.

(up, i). According to the invariant, the stack contains a valuation $x_1,\ldots,x_i$ that makes the formula $(Q_{i+1}x_{i+1})\ldots(Q_nx_n)\Phi$ true. If Q_i is $\exists$, then the current value of x_i is the witness that shows that $x_1,\ldots,x_{i-1}$ makes $(Q_ix_i)(Q_{i+1}x_{i+1})\ldots(Q_nx_n)\Phi$ true. So pop the stack and go to state (up, $i-1$).

If Q_i is $\forall$ and $x_i = 0$, this is the "infix" visit to this node: set $x_i = 1$ (by changing the topmost cell on the stack from 0 to 1), and go to the state (down, $i+1$). If Q_i is $\forall$ and $x_i = 1$, this is the "postfix" visit to this node; at this point we have succeeded in evaluating the formula at this node, so go to state (up, $i-1$).

Hence if we reach the state (up, 0), the stack will be empty and the original formula must have been true. Furthermore, it is clear that the machine M explores the entire subtree, so if the formula is true, all the needed witnesses will be found. □

Theorem 7 *The ARPDA termination problem is polytime reducible to* C-REG-SAT *over any nontrivial poset C.*

Proof: Let $M = (Q, q^0, q^f, \Delta)$ be an ARPDA. We denote the initial and final states with superscripts to avoid conflicts with the subscripted variables of *C-REG-SAT*. Let C be a non-trivial poset with $a \leq b$, $a \neq b$ holding in C. We will construct an instance Σ_M of *C-REG-SAT* such that Σ_M is unsatisfiable iff M halts.

The variables of Σ_M are Q. The inequalities are

$$p_{wa} \leq q_{wb}$$

for every $w \in \{0,1\}^*$ and $((p,a) \mapsto (q,b)) \in \Delta$,

$$p_w \leq q_w$$

for $((p,R) \mapsto q) \in \Delta$ and $w \in R$, and the two inequalities

$$b \leq q^0, \quad q^f \leq a$$

It is clear that in M, (p,w) reduces to (q,u) in at most k steps iff the assertion $p_w \leq q_u$ is deducible from Σ_M in at most k applications of transitivity. Hence M halts iff $q^0 \leq q^f$ is deducible from Σ_M.

We claim that M halts iff Σ_M is unsatisfiable. Assume M halts and Σ_M is satisfiable with solution σ. Then we have $b \leq \sigma(q^0) \leq \sigma(q^f) \leq a$, so $a = b$, contradicting our assumption that $a \neq b$.

If M does not halt, construct a solution σ to Σ_M as follows: If (q,w) is reachable from the initial state (q^0,ϵ), assign $\sigma(q_w) = b$. Otherwise assign $\sigma(q_w) = a$. It is easy to show that this assignment satisfies all the inequalities in Σ_M. □

It should be noted that *C-REG-SAT* exhibits dramatically different behavior than its fragment *C-FIN-SAT* which consists of finite instances of *C-REG-SAT*. As we remarked earlier, *C-FIN-SAT* is always in NP. It follows from the results of [11] that there are finite posets for which *C-FIN-SAT* is actually NP-complete. Our results of this paper indicate that *C-REG-SAT* is always between *PSPACE* and *DEXPTIME*, for all posets C which are not discrete. Over discrete C, *C-REG-SAT* is clearly in *PTIME*.

9 Conclusions

We have shown how to extend Mitchell's algorithm for type reconstruction in a type system with atomic subtyping to handle recursive types. This extension is necessary to do type reconstruction for object-oriented systems with **self**. The resulting algorithm is in *DEXPTIME*, which also improves the previous *NEXPTIME* algorithm for atomic subtyping on finite types.

References

[1] Roberto M. Amadio and Luca Cardelli. Subtyping Recursive Types. In *Conf. Rec. 18th ACM Symposium on Principles of Programming Languages*, pages 104–118, 1991.

[2] Kim B. Bruce. A Paradigmatic Object-Oriented Programming Language: Design, Static Typing and Semantics. Technical Report CS-92-01, Williams College, January 1992.

[3] William R. Cook, Walter L. Hill, and Peter S. Canning. Subtyping is not Inheritance. In *Conf. Rec. 17th ACM Symposium on Principles of Programming Languages*, pages 125–135, 1990.

[4] Bruno Courcelle. Fundamental Properties of Infinite Trees. *Theoretical Computer Science*, 25:95–169, 1983.

[5] Y.-C. Fuh and P. Mishra. Type Inference with Subtypes. In *Proceedings European Symposium on Programming*, pages 94–114, 1988.

[6] Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient Inference of Partial Types. Technical Report DAIMI PB-394, Computer Science Department, Aarhus University, April 1992.

[7] Patrick Lincoln and John C. Mitchell. Algorithmic Aspects of Type Inference with Subtypes. In *Conf. Rec. 19th ACM Symposium on Principles of Programming Languages*, pages 293–304, 1992.

[8] John C. Mitchell. Coercion and Type Inference (summary). In *Conf. Rec. 11th ACM Symposium on Principles of Programming Languages*, pages 175–185, 1984.

[9] John C. Mitchell. Type Inference with Simple Subtypes. *Journal of Functional Programming*, 1:245–285, 1991.

[10] Patrick M. O'Keefe and Mitchell Wand. Type Inference for Partial Types is Decidable. In Bernd Krieg-Brückner, editor, *European Symposium on Programming '92*, volume 582 of *Springer Lecture Notes in Computer Science*, pages 408–417. Springer-Verlag, 1992.

[11] Vaughn Pratt and Jerzy Tiuryn. Satisfiability of Inequalities in a Poset. to appear, 1992.

[12] Michael O. Rabin. Weakly Definable Relations and Special Automata. In Y. Bar-Hillel, editor, *Mathematical Logic and the Foundations of Set Theory*, pages 1–23, Amsterdam, 1970. North-Holland.

[13] Satish Thatte. Type Inference with Partial Types. In *Proceedings International Colloquium on Automata, Languages, and Programming '88*, pages 615–629, 1988.

[14] Jerzy Tiuryn. Subtype Inequalities. In *Proc. 7th IEEE Symposium on Logic in Computer Science*, pages 308–315, 1992.

[15] Moshe Y. Vardi and Pierre Wolper. Automata-Theoretic techniques for modal logics of programs. *J. Comp. Sys. Sci.*, 32:183–221, 1986.

[16] Mitchell Wand. A Simple Algorithm and Proof for Type Inference. *Fundamenta Informaticae*, 10:115–122, 1987.

[17] Mitchell Wand and Patrick M. O'Keefe. On the Complexity of Type Inference with Coercion. In *Conf. on Functional Programming Languages and Computer Architecture*, 1989.