Mechanizing Logical Relations*

Allen Stoughton†
Department of Computing and Information Sciences
Kansas State University
Manhattan, KS 66506, USA
E-mail: `allen@cis.ksu.edu`

Abstract. We give an algorithm for deciding whether there exists a definable element of a finite model of an applied typed lambda calculus that passes certain tests, in the special case when all the constants and test arguments are of order at most one. When there is such an element, the algorithm outputs a term that passes the tests; otherwise, the algorithm outputs a logical relation that demonstrates the nonexistence of such an element. Several example applications of the C implementation of this algorithm are considered.

1 Introduction

Given a model of an applied typed lambda calculus, it is natural to consider the problem of determining whether an element of that model is definable by a term, or, more generally, of determining whether there exists a definable element of the model that passes certain tests. One approach to settling such questions makes use of so-called "logical relations" [Plo80].

Building on recent work on logical relations by Sieber [Sie92], we give an algorithm for deciding whether there exists a definable element of a finite model that passes certain tests, in the special case when all the constants and test arguments are of order at most one. When there is such an element, the algorithm outputs a term that passes the tests; otherwise, the algorithm outputs a logical relation that demonstrates the nonexistence of such an element. Loader's recent proof of the undecidability of the lambda definability problem [Loa94] shows that the restriction to constants and test arguments of order at most one is necessary. (Specifically, Loader shows the undecidability of the problem of determining the definability of order-three elements of the full type hierarchy over a seven element set.)

The algorithm was first implemented in Standard ML and used to find an interesting non-definability proof (see Lemma 4.16 of [JS93]). An efficient implementation of the algorithm in ANSI C has now been written and applied to various definability problems, some examples of which are described below. A copy of this program, *lambda*, along with supporting documentation and a number of example lambda definability problems, can be obtained by anonymous ftp. Connect to `ftp.cis.ksu.edu`, login as `anonymous`, change directory to `pub/CIS/Stoughton/lambda`, retrieve the file `README`, and follow the instructions given in that file.

2 The typed lambda calculus

This section consists of the mostly standard definitions concerning the syntax and semantics of the typed lambda calculus that will be required in the sequel. An introduction to the typed lambda

*A corrected version of the paper that appears in *Ninth International Conference on the Mathematical Foundations of Programming Semantics*, Lecture Notes in Computer Science, vol. 802, pp. 359–377, Springer-Verlag, 1994. (See the footnote on p. 2 for the single correction.)

†The research reported here was partially supported by ESPRIT project CLICS-II and was performed while the author was on the faculty of the School of Cognitive and Computing Sciences of the University of Sussex.

calculus can be found, e.g., in [Mit90].

The set of *types* T is least such that

(i) $\iota \in T$,

(ii) $\sigma \to \tau \in T$ if $\sigma \in T$ and $\tau \in T$.

We let $\to$ associate to the right. The *order* $\operatorname{ord}\sigma \in \omega$ of a type $\sigma \in T$ is defined by $\operatorname{ord}\iota = 0$ and $\operatorname{ord}(\sigma\to\tau) =$ the maximum of $1+\operatorname{ord}\sigma$ and $\operatorname{ord}\tau$. The *arity* $\operatorname{ar}\sigma \in \omega$ of a type σ is defined by $\operatorname{ar}\iota = 0$ and $\operatorname{ar}(\sigma\to\tau) = 1+\operatorname{ar}\tau$. Thus, if $n > 0$ and $\sigma_i \in T$ for all $i \in n$, then $\operatorname{ord}(\sigma_0\to\cdots\to\sigma_{n-1}\to\iota) = 1+$ the maximum of $\{\operatorname{ord}\sigma_i \mid i \in n\}$ and $\operatorname{ar}(\sigma_0 \to \cdots \to \sigma_{n-1} \to \iota) = n$.

Define σ^n, for $n \in \omega$, by: $\sigma^0 = \sigma$ and $\sigma^{n+1} = \sigma \to \sigma^n$. Thus, for all $n \in \omega$, $\operatorname{ar}\sigma^n = n + \operatorname{ar}\sigma$ and $\operatorname{ord}\sigma^n$ is $\operatorname{ord}\sigma$, if $n = 0$, and is $1 + \operatorname{ord}\sigma$, otherwise. It is easy to see that σ has order at most one just when it is of the form ι^n for some $n \in \omega$.

Many operations and concepts extend naturally from sets to T-indexed families of sets, in a pointwise manner. For example, given an ordinal α, an *α-ary relation* $R_{(-)}$ over a T-indexed family of sets $A_{(-)}$ is a T-indexed family of α-ary relations R_σ over A_σ. We will make use of this and other such extensions without explicit comment. We sometimes confuse a T-indexed family of sets A with $\bigcup_{\sigma\in T} A_\sigma$.

V is a T-indexed family of disjoint, denumerable sets of *variables*. A *family of constants* C is a T-indexed family of disjoint sets. We say that such a C is *finite* iff $\bigcup_{\sigma\in T} C_\sigma$ is finite, and that C is *infinite* otherwise. The *order* $\operatorname{ord}C \in \omega \cup \{\infty\}$ of C is the greatest element of $\{\operatorname{ord}\sigma \mid \sigma \in T$ and $C_\sigma \neq \emptyset\}$ if it exists, and ∞ otherwise.

The family $\Lambda(C)$ of *typed λ-terms* over a family of constants C is least such that

(i) $c \in \Lambda(C)_\sigma$ if $c \in C_\sigma$,

(ii) $x \in \Lambda(C)_\sigma$ if $x \in V_\sigma$,

(iii) $M\,N \in \Lambda(C)_\tau$ if $M \in \Lambda(C)_{\sigma\to\tau}$ and $N \in \Lambda(C)_\sigma$,

(iv) $\lambda x.\,M \in \Lambda(C)_{\sigma\to\tau}$ if $x \in V_\sigma$ and $M \in \Lambda(C)_\tau$.

We call a term $M\,N$ an *application* and a term $\lambda x.\,M$ an *abstraction*. We let application associate to the left, and abbreviate $\lambda x_0.\cdots\lambda x_{n-1}.\,M$ to $\lambda x_0\cdots x_{n-1}.\,M$. (When $n = 0$, $\lambda x_0\cdots x_{n-1}.\,M = M$.) The set of *free variables* $\operatorname{fv}M \in \mathcal{P}(\bigcup_{\sigma\in T} V_\sigma)$ of a term $M \in \Lambda(C)$ is defined by $\operatorname{fv}c = \emptyset$, $\operatorname{fv}x = \{x\}$, $\operatorname{fv}(M\,N) = \operatorname{fv}M \cup \operatorname{fv}N$ and $\operatorname{fv}(\lambda x.\,M) = \operatorname{fv}M - \{x\}$. A term $M \in \Lambda(C)$ is *closed* iff $\operatorname{fv}M = \emptyset$, and *open* otherwise.

We write $\Gamma(C)$ for the family of *λ-free terms* over C: $\Gamma(C)_\sigma = \{M \in \Lambda(C)_\sigma \mid M$ is λ-free$\}$. The *depth* $\operatorname{depth}M \in \omega$ of a λ-free term M is defined by $\operatorname{depth}c = \operatorname{depth}x = 0$ and $\operatorname{depth}(M\,N) =$ the maximum of $\operatorname{depth}M$ and $1 + \operatorname{depth}N$. The *size* $\operatorname{size}M \in \omega$ of a λ-free term M is defined by $\operatorname{size}c = \operatorname{size}x = 1$ and $\operatorname{size}(M\,N) = \operatorname{size}M + \operatorname{size}N$. Thus, if $n > 0$, $\sigma_i \in T$ for all $i \in n$, $M_i \in \Gamma(C)_{\sigma_i}$ for all $i \in n$ and d is a constant or variable of type $\sigma_0\to\cdots\to\sigma_{n-1}\to\iota$, then $\operatorname{depth}(d\,M_0\,\cdots\,M_{n-1}) = 1+$ the maximum of $\{\operatorname{depth}M_i \mid i \in n\}$ and $\operatorname{size}(d\,M_0\,\cdots\,M_{n-1}) = 1 + \operatorname{size}M_0 + \cdots + \operatorname{size}M_{n-1}$.

We write $f\,a$ for the application of a function f to an argument a, and let function application associate to the left. The set of all functions from a set A to a set B is denoted by $A \to B$, and $\to$ associates to the right.

A *type frame* A is a T-indexed set such that $A_\sigma \neq \emptyset$ for all $\sigma \in T$ [1] and $A_{\sigma\to\tau} \subseteq A_\sigma \to A_\tau$ for all $\sigma, \tau \in T$. We say that such an A is *finite* iff A_ι is finite, and that A is *infinite* otherwise. The set Env_A (or just Env) of *environments* over A consists of the set of all type-respecting functions from $\bigcup_{\sigma\in T} V_\sigma$ to $\bigcup_{\sigma\in T} A_\sigma$. If $\rho \in \mathrm{Env}$, $a \in A_\sigma$ and $x \in V_\sigma$, then $\rho[a/x] \in \mathrm{Env}$ is the environment that

[1] This reads $A_\iota \neq \emptyset$ in the published version of the paper.

sends x to a, and sends all $y \neq x$ to $\rho\, y$. We write Sem_A (or just Sem) for the T-indexed family of sets defined by $\mathrm{Sem}_\sigma = \mathrm{Env} \to A_\sigma$.

A $\Lambda(C)$-*model* $\mathcal{A}$ consists of a type frame A, together with an element $c_\mathcal{A} \in A_\sigma$ for each $c \in C_\sigma$, such that the following recursive definition of the meaning $[\![M]\!] \in \mathrm{Sem}_\sigma$ of a term $M \in \Lambda(C)_\sigma$ is well-defined:

$$\begin{aligned}
[\![c]\!]\rho &= c_\mathcal{A} \\
[\![x]\!]\rho &= \rho\, x \\
[\![M\, N]\!]\rho &= ([\![M]\!]\rho)([\![N]\!]\rho) \\
[\![\lambda x.\, M]\!]\,\rho\, a &= [\![M]\!]\rho[a/x].
\end{aligned}$$

When M is closed, we often write $[\![M]\!]$ for $[\![M]\!]\rho$, where $\rho \in \mathrm{Env}$ is arbitrary. An element $a \in A_\sigma$ is *definable* iff there exists a closed term $M \in \Lambda(C)_\sigma$ such that $a = [\![M]\!]$. We say that $\mathcal{A}$ is *finite* iff A is finite, and that $\mathcal{A}$ is *infinite* otherwise.

Our example model in the sequel will be the *monotone function model* of *Finitary PCF*: the restriction of PCF [Plo77] to the booleans. We write FPCF for the family of constants such that $\mathrm{FPCF}_\iota = \{\Omega, \mathrm{tt}, \mathrm{ff}\}$, $\mathrm{FPCF}_{\iota^3} = \mathrm{If}$, and $\mathrm{FPCF}_\sigma = \emptyset$ for all other $\sigma \in T$, and define a finite $\Lambda(\mathrm{FPCF})$-model $\mathcal{F}$ as follows. F_ι is the poset $\{\bot, \mathrm{tt}, \mathrm{ff}\}$, where $\bot$ is $\sqsubseteq$ the incomparable elements tt and ff, and $F_{\sigma\to\tau}$ is the set of all monotonic functions from F_σ to F_τ, ordered pointwise ($f \sqsubseteq g$ iff $f\, a \sqsubseteq g\, a$ for all a). We then set $\Omega_\mathcal{F} = \bot$, $\mathrm{tt}_\mathcal{F} = \mathrm{tt}$, $\mathrm{ff}_\mathcal{F} = \mathrm{ff}$ and define $\mathrm{If}_\mathcal{F}$ by

$$\mathrm{If}_\mathcal{F}\; x\, y\, z = \begin{cases} \bot & \text{if } x = \bot, \\ y & \text{if } x = \mathrm{tt}, \\ z & \text{if } x = \mathrm{ff}. \end{cases}$$

One shows that the meaning function for $\mathcal{F}$ is well-defined by ordering Env_F pointwise and showing by induction on M that $[\![M]\!]$ is both well-defined and monotonic.

3 Definability

We now consider the problem of determining whether an element of a $\Lambda(C)$-model is definable, or, more generally, of determining whether there exists a definable element of a $\Lambda(C)$-model that passes certain tests. For example, we can ask whether the "parallel or" operation of the $\Lambda(\mathrm{FPCF})$-model $\mathcal{F}$ is definable, i.e., whether there exists a closed term M of type ι^2 such that

$$\begin{aligned}
[\![M]\!] \quad \mathrm{tt} \quad \bot &= \mathrm{tt} \\
[\![M]\!] \quad \bot \quad \mathrm{tt} &= \mathrm{tt} \\
[\![M]\!] \quad \mathrm{ff} \quad \mathrm{ff} &= \mathrm{ff}.
\end{aligned}$$

One approach to settling such questions makes use of so-called "logical relations" [Plo80]. It is easier to say what logical relations are if we first extend function application from elements of type frames to tuples of elements of type frames, in a componentwise manner. Suppose A is a type frame, α is an ordinal and $\sigma, \tau \in T$. If $X = \langle\, x_\lambda \in A_{\sigma\to\tau} \mid \lambda \in \alpha \,\rangle$ and $Y = \langle\, y_\lambda \in A_\sigma \mid \lambda \in \alpha \,\rangle$, then we define the *application* $X\,Y$ of X to Y to be $\langle\, x_\lambda\, y_\lambda \in A_\tau \mid \lambda \in \alpha \,\rangle$, and let $X\,Y$ associate to the left. Given an $a \in A_\sigma$, we sometimes write a for $\langle\, a \mid \lambda \in \alpha \,\rangle \in A_\sigma^\alpha$.

An α*-ary logical relation* R over a type frame A is an α-ary relation over A such that $X \in R_{\sigma\to\tau}$ iff $X\,Y \in R_\tau$ for all $Y \in R_\sigma$. We say that an α-tuple $X \in A^\alpha_\sigma$ *satisfies* such an R iff $X \in R_\sigma$. An α*-ary logical relation* R over a $\Lambda(C)$-model $\mathcal{A}$ is an α-ary logical relation over A such that $c_{\mathcal{A}}$ satisfies R for all $c \in C$.

The following theorem and its corollary show why logical relations are useful for showing non-definability results.

Theorem 3.1 (Plotkin) *If R is an α-ary logical relation over a $\Lambda(C)$-model $\mathcal{A}$, then $[\![M]\!]$ satisfies R for all closed $M \in \Lambda(C)$.*

Proof. An easy induction on $\Lambda(C)$ shows that, for all $M \in \Lambda(C)_\sigma$ and $\rho_\lambda \in \mathrm{Env}$ for all $\lambda \in \alpha$, if $\langle\, \rho_\lambda\, x \mid \lambda \in \alpha \,\rangle \in R_\tau$ for all $x \in \mathrm{fv}\, M \cap V_\tau$ and $\tau \in T$, then $\langle\, [\![M]\!]\rho_\lambda \mid \lambda \in \alpha \,\rangle \in R_\sigma$. The result then follows immediately. □

Corollary 3.2 *Let $\mathcal{A}$ be a $\Lambda(C)$-model, $\Delta_i \in A^\alpha_{\sigma_i}$ for all $i \in m$, $X \in A^\alpha_\iota$ and R be an α-ary logical relation over $\mathcal{A}$, for $m \in \omega$ and an ordinal α. If R is satisfied by Δ_i for all $i \in m$ but is not satisfied by X, then there is no definable $a \in A_{\sigma_0\to\cdots\to\sigma_{m-1}\to\iota}$ such that $a\,\Delta_0 \cdots \Delta_{m-1} = X$.*

Proof. Immediate from Theorem 3.1. □

We can, e.g., use Corollary 3.2 to prove Plotkin's result [Plo77] that parallel or is not definable in Finitary PCF. (Although the following proof is due to Plotkin, he never published it. It was recently rediscovered by Sieber [Sie92].) Define *argument tuples* $\Delta_i \in F^3_\iota$ for all $i \in 2$ and a *result tuple* $X \in F^3_\iota$ by taking $\Delta_0 = \langle \mathrm{tt}, \bot, \mathrm{ff} \rangle$ (the first argument column of the display at the beginning of this section), $\Delta_1 = \langle \bot, \mathrm{tt}, \mathrm{ff} \rangle$ (the second argument column of that display) and $X = \langle \mathrm{tt}, \mathrm{tt}, \mathrm{ff} \rangle$ (the result column of that display). Let R be the ternary logical relation over F such that $\langle x, y, z \rangle \in R_\iota$ iff $x = y = z$ or one of x or y is $\bot$. It is easy to show that R is satisfied by the interpretations of Ω, tt, ff and If. But R is satisfied by Δ_0 and Δ_1 but not by X, allowing us to conclude that there is no definable $f \in F_{\iota^2}$ such that $f\,\Delta_0\,\Delta_1 = X$.

Loader's recent proof of the undecidability of the lambda definability problem [Loa94] shows that Corollary 3.2 fails to provide a complete method for showing non-definability (and thus definability) results. However, a slight generalization of Theorem 4.1 of [Sie92] shows that it does provide a complete method in the special case where the orders of C and the σ_i's are at most one (cf., Theorem 1 of [Plo80] and Theorem 5 of [JT93]).

Definition 3.3 Suppose $\mathcal{A}$ is a $\Lambda(C)$-model and $\Delta = \langle\, \Delta_i \in A^\alpha_{\sigma_i} \mid i \in m \,\rangle$, for $m \in \omega$ and an ordinal α and where C and the σ_i's have order at most one. Then, $R(\Delta)$ is the α-ary logical relation over A such that $X \in R(\Delta)_\iota$ iff $a\,\Delta_0 \cdots \Delta_{m-1} = X$ for some definable $a \in A_{\sigma_0\to\cdots\to\sigma_{m-1}\to\iota}$.

Lemma 3.4 (Sieber) *Suppose $\mathcal{A}$ is a $\Lambda(C)$-model and $\Delta = \langle\, \Delta_i \in A^\alpha_{\sigma_i} \mid i \in m \,\rangle$, for $m \in \omega$ and an ordinal α and where C and the σ_i's have order at most one. Then, $R(\Delta)$ is an α-ary logical relation over $\mathcal{A}$ that is satisfied by Δ_i for all $i \in m$.*

Proof. Suppose that $c \in C_{\iota^n}$. If $Y_0, \ldots, Y_{n-1} \in R(\Delta)_\iota$, then there are closed terms $M_0, \ldots, M_{n-1}$ of type $\sigma_0 \to \cdots \to \sigma_{m-1} \to \iota$ such that $[\![M_j]\!]\,\Delta_0 \cdots \Delta_{m-1} = Y_j$ for all $j \in n$. Then, the term

$$M = \lambda x_0 \cdots x_{m-1}.\, c\,(M_0\, x_0 \cdots x_{m-1}) \cdots (M_{n-1}\, x_0 \cdots x_{m-1})$$

of type $\sigma_0 \to \cdots \to \sigma_{m-1} \to \iota$ is such that

$$[\![M]\!]\, \Delta_0 \cdots \Delta_{m-1} = c_{\mathcal{A}}\, Y_0 \cdots Y_{n-1},$$

showing that $c_{\mathcal{A}}\, Y_0 \cdots Y_{n-1} \in R(\Delta)_\iota$. Thus $c_{\mathcal{A}}$ satisfies $R(\Delta)$. The proof that Δ_i satisfies $R(\Delta)$ for all $i \in m$ is almost identical (x_i is used in the term M instead of c). $\square$

Theorem 3.5 (Sieber) *Suppose $\mathcal{A}$ is a $\Lambda(C)$-model, $\Delta = \langle\, \Delta_i \in A^\alpha_{\sigma_i} \mid i \in m \,\rangle$ and $X \in A^\alpha_\iota$, for $m \in \omega$ and an ordinal α and where C and the σ_i's have order at most one. Then, $a\, \Delta_0 \cdots \Delta_{m-1} = X$ for some definable $a \in A_{\sigma_0 \to \cdots \to \sigma_{m-1} \to \iota}$ iff every α-ary logical relation over $\mathcal{A}$ that is satisfied by Δ_i for all $i \in m$ is also satisfied by X.*

Proof. Immediate from Corollary 3.2 and Lemma 3.4. $\square$

Although Theorem 3.5 gives a characterization of $R(\Delta)_\iota$, the fact that this characterization involves the universal quantification over all α-ary logical relations over $\mathcal{A}$ that are satisfied by the Δ_i limits its practical utility. It turns out, however, that we can give a much more direct characterization of $R(\Delta)_\iota$.

Definition 3.6 Suppose $\mathcal{A}$ is a $\Lambda(C)$-model and $\Delta = \langle\, \Delta_i \in A^\alpha_{\sigma_i} \mid i \in m \,\rangle$, for $m \in \omega$ and an ordinal α and where C and the σ_i's have order at most one. Then, $L(\Delta)$ is the α-ary logical relation over A such that $L(\Delta)_\iota$ is the least α-ary relation over A_ι that is closed under $c_{\mathcal{A}}$, for all $c \in C$, and Δ_i, for all $i \in m$, where the $c_{\mathcal{A}}$'s and Δ_i's are viewed as operations over A^α_ι in the obvious way.

Lemma 3.7 *Suppose $\mathcal{A}$ is a $\Lambda(C)$-model and $\Delta = \langle\, \Delta_i \in A^\alpha_{\sigma_i} \mid i \in m \,\rangle$, for $m \in \omega$ and an ordinal α and where C and the σ_i's have order at most one. Let $x_i \in V_{\sigma_i}$ for all $i \in m$ be distinct variables.*

(i) *Suppose that $c \in C_{\iota^n}$, $Y_0, \ldots, Y_{n-1} \in A^\alpha_\iota$ and $M_0, \ldots, M_{n-1} \in \Gamma(C)_\iota$. If $\operatorname{fv} M_j \subseteq \{x_0, \ldots, x_{m-1}\}$ and*

$$[\![\lambda x_0 \cdots x_{m-1}.\, M_j]\!]\, \Delta_0 \cdots \Delta_{m-1} = Y_j,$$

for all $j \in n$, then the λ-free term $M = c\, M_0 \cdots M_{n-1}$ of type ι is such that $\operatorname{fv} M \subseteq \{x_0, \ldots, x_{m-1}\}$ and

$$[\![\lambda x_0 \cdots x_{m-1}.\, M]\!]\, \Delta_0 \cdots \Delta_{m-1} = c_{\mathcal{A}}\, Y_0 \cdots Y_{n-1}.$$

(ii) *Suppose that $i \in m$, $Y_0, \ldots, Y_{\operatorname{ar}\sigma_i - 1} \in A^\alpha_\iota$ and $M_0, \ldots, M_{\operatorname{ar}\sigma_i - 1} \in \Gamma(C)_\iota$. If $\operatorname{fv} M_j \subseteq \{x_0, \ldots, x_{m-1}\}$ and*

$$[\![\lambda x_0 \cdots x_{m-1}.\, M_j]\!]\, \Delta_0 \cdots \Delta_{m-1} = Y_j,$$

for all $j \in \operatorname{ar}\sigma_i$, then the λ-free term $M = x_i\, M_0 \cdots M_{\operatorname{ar}\sigma_i - 1}$ of type ι is such that $\operatorname{fv} M \subseteq \{x_0, \ldots, x_{m-1}\}$ and

$$[\![\lambda x_0 \cdots x_{m-1}.\, M]\!]\, \Delta_0 \cdots \Delta_{m-1} = \Delta_i\, Y_0 \cdots Y_{\operatorname{ar}\sigma_i - 1}.$$

(iii) *For all $X \in L(\Delta)_\iota$, there is an $M \in \Gamma(C)_\iota$ such that $\operatorname{fv} M \subseteq \{x_0, \ldots, x_{m-1}\}$ and*

$$[\![\lambda x_0 \cdots x_{m-1}.\, M]\!]\, \Delta_0 \cdots \Delta_{m-1} = X.$$

Proof. (i) and (ii) are immediate, and (iii) follows from (i) and (ii) by induction on $L(\Delta)_\iota$. $\square$

Lemma 3.8 *Suppose $\mathcal{A}$ is a $\Lambda(C)$-model and $\Delta = \langle\, \Delta_i \in A^{\alpha}_{\sigma_i} \mid i \in m \,\rangle$, for $m \in \omega$ and an ordinal α and where C and the σ_i's have order at most one. Then, $L(\Delta) = R(\Delta)$.*

Proof. $L(\Delta)$ is clearly an α-ary logical relation over $\mathcal{A}$ that is satisfied by Δ_i for all $i \in m$, and $L(\Delta)_\iota \subseteq R(\Delta)_\iota$ follows from Lemma 3.7 (iii). For the opposite inclusion, if $X \notin L(\Delta)_\iota$, then there is no definable $a \in A$ such that $a\,\Delta_0 \cdots \Delta_{m-1} = X$, by Corollary 3.2, and thus $X \notin R(\Delta)_\iota$. $\square$

Theorem 3.9 *Suppose $\mathcal{A}$ is a $\Lambda(C)$-model, $\Delta = \langle\, \Delta_i \in A^{\alpha}_{\sigma_i} \mid i \in m \,\rangle$ and $X \in A^{\alpha}_{\iota}$, for $m \in \omega$ and an ordinal α and where C and the σ_i's have order at most one. Then, $a\,\Delta_0 \cdots \Delta_{m-1} = X$ for some definable $a \in A_{\sigma_0 \to \cdots \to \sigma_{m-1} \to \iota}$ iff $X \in L(\Delta)_\iota$.*

Proof. Immediate from Lemma 3.8. $\square$

Theorem 3.9 and Lemma 3.7 suggest the following algorithm schema.

Algorithm Schema 3.10 *Inputs.* A finite family of constants C of order at most one, $m, \alpha \in \omega$, types $\sigma_0, \ldots, \sigma_{m-1}$ of order at most one, a finite, nonempty set A_ι, $c_A \in A_{\iota^n}$ for each $c \in C_{\iota^n}$, $\Delta = \langle\, \Delta_i \in A^{\alpha}_{\sigma_i} \mid i \in m \,\rangle$ and $X \in A^{\alpha}_{\iota}$, where we extend A_ι to a type frame A by taking $A_{\sigma \to \tau}$ to be the set of all functions from A_σ to A_τ for all $\sigma, \tau \in T$.

Initialization. Pick distinct variables $x_i \in V_{\sigma_i}$ for all $i \in m$. Initialize the *stage* $k \in \omega$ to 0. Let $Z \subseteq U$ be

$$\{\, \langle c_A,\, c\rangle \mid c \in C_\iota \,\} \cup \{\, \langle \Delta_i,\, x_i\rangle \mid i \in m \text{ and } \sigma_i = \iota \,\},$$

where U is set of all pairs $\langle Y, M\rangle$ such that $Y \in A^{\alpha}_{\iota}$, $M \in \Gamma(C)_\iota$ and $\mathrm{fv}\, M \subseteq \{x_0, \ldots, x_{m-1}\}$. Initialize the *state* $S \subseteq U$ to a subset of Z that is a function with domain $\mathrm{dom}\, Z$. (The particular subset chosen is left unspecified, as is the method used to compute that subset; it need not involve the construction of Z.) If $\langle X, M\rangle \in S$ for some term M, then terminate with k and the term $\lambda x_0 \cdots x_{m-1}.\, M$.

Loop. Let $Z = Z_1 \cup Z_2$, where $Z_1 \subseteq U$ is the set of all

$$\langle c_A\, Y_0\, \cdots\, Y_{n-1},\; c\, M_0\, \cdots\, M_{n-1}\rangle$$

such that $c \in C_{\iota^n}$, $n > 0$ and $\langle Y_j, M_j\rangle \in S$ for all $j \in n$, and $Z_2 \subseteq U$ is the set of all

$$\langle \Delta_i\, Y_0\, \cdots\, Y_{\mathrm{ar}\,\sigma_i - 1},\; x_i\, M_0\, \cdots\, M_{\mathrm{ar}\,\sigma_i - 1}\rangle$$

such that $i \in m$, $\mathrm{ar}\,\sigma_i > 0$ and $\langle Y_j, M_j\rangle \in S$ for all $j \in \mathrm{ar}\,\sigma_i$. Pick a subset S' of Z such that S' is a function with domain $\mathrm{dom}\, Z - \mathrm{dom}\, S$ and (†) $\langle Y, M\rangle \in S'$ implies that $\mathrm{size}\, M \leq \mathrm{size}\, N$ for all N that are paired with Y in Z. (The particular subset chosen is left unspecified, as is the method used to compute that subset; it need not involve the construction of Z, Z_1 or Z_2.) If $S' = \emptyset$, then terminate with k and $\mathrm{dom}\, S$. Otherwise, increment k by one and add the elements of S' to S. (‡) If $\langle X, M\rangle \in S$ for some term M, then terminate with k and the term $\lambda x_0 \cdots x_{m-1}.\, M$. Otherwise, repeat.

An *instance* of Algorithm Schema 3.10 is an algorithm formed from the schema by specifying the details that were left open. Condition (†) is included since experience suggests that this will ensure that instances of the schema will generate good quality terms. Theorem 3.11 doesn't depend upon (†) being included, however.

Theorem 3.11 *If we supply the required inputs to an instance of Algorithm Schema 3.10, then one of the following statements holds.*

(i) *The algorithm terminates with a stage l and a closed term of the form $\lambda x_0 \cdots x_{m-1}.M$, for distinct variables $x_i \in V_{\sigma_i}$ and a λ-free term M of type ι and depth l. Let $\mathcal{B}$ be any $\Lambda(C)$-model such that $B_\iota = A_\iota$, $c_{\mathcal{B}} = c_A$ for all $c \in C$, and $\Delta_i \in B^\alpha_{\sigma_i}$ for all $i \in m$. Then, $[\![\lambda x_0 \cdots x_{m-1}.\, M]\!]\, \Delta_0 \cdots \Delta_{m-1} = X$. Furthermore, if $N \in \Gamma(C)_\iota$ is such that $\mathrm{fv}\, N \subseteq \{x_0, \ldots, x_{m-1}\}$ and $[\![\lambda x_0 \cdots x_{m-1}.\, N]\!]\, \Delta_0 \cdots \Delta_{m-1} = X$, then $\mathrm{depth}\, M \leq \mathrm{depth}\, N$.*

(ii) *The algorithm terminates with a stage l and an α-ary relation Q over A_ι such that $X \notin Q$. If $\mathcal{B}$ is a $\Lambda(C)$-model with the above properties, then $Q = L(\Delta)_\iota$, so that there is no definable $b \in B$ such that $b\, \Delta_0 \cdots \Delta_{m-1} = X$.*

Proof. Let S_0 be the initial value of S, and S_l, for $l \geq 1$, be S's value when point (‡) is reached for the lth time (at which point k's value will be l; S_l is undefined if the algorithm terminates before (‡) has been executed l times). Then, the following properties hold (for (d)–(f), $\mathcal{B}$ is a $\Lambda(C)$-model with the properties specified in the theorem's statement):

(a) If S_l is defined, then S_l is a function.

(b) If S_{l+1} is defined, then it is a proper superset of S_l.

(c) If S_l is defined, $\langle Y, M\rangle \in S_l$ and either $l = 0$ or $Y \notin \mathrm{dom}\, S_{l-1}$, then $\mathrm{depth}\, M = l$.

(d) If S_l is defined, then $\mathrm{dom}\, S_l \subseteq L(\Delta)_\iota$.

(e) If S_l is defined and $\langle Y, M\rangle \in S_l$, then $[\![\lambda x_0 \cdots x_{m-1}.\, M]\!]\, \Delta_0 \cdots \Delta_{m-1} = Y$.

(f) If S_l is defined, $M \in \Gamma(C)_\iota$, $\mathrm{fv}\, M \subseteq \{x_0, \ldots, x_{m-1}\}$ and $\mathrm{depth}\, M = l$, then $[\![\lambda x_0 \cdots x_{m-1}.\, M]\!]\, \Delta_0 \cdots \Delta_{m-1} \in \mathrm{dom}\, S_l$.

The proofs of properties (a), (d) and (e) are by induction on l, and Lemma 3.7 (i) and (ii) are used in (e)'s proof. The proof of (b) is obvious.

For (c), we use a course of values induction on l. We consider the case where M has the form $c\, M_0 \cdots M_{n-1}$ (the case where M has the form $x_i\, M_0 \cdots M_{\mathrm{ar}\,\sigma_i - 1}$ is similar). If $l = 0$, then $n = 0$, and thus $\mathrm{depth}\, M = \mathrm{depth}\, c = 0$. So, suppose that $l > 0$, so that $Y \notin \mathrm{dom}\, S_{l-1}$. Then, $n > 0$ and there are $Y_j \in A^\alpha_\iota$ for all $j \in n$ such that $\langle Y_j, M_j\rangle \in S_{l-1}$ for all $j \in n$ and $Y = c_A\, Y_0 \cdots Y_{n-1}$. Let the stages $p_j < l$ for all $j \in n$ be such that $Y_j \in \mathrm{dom}\, S_{p_j}$ and either $p_j = 0$ or $Y_j \notin \mathrm{dom}\, S_{p_j - 1}$. Then, $\mathrm{depth}\, M_j = p_j$ for all $j \in n$, by the inductive hypotheses for the p_j's, so that $\mathrm{depth}\, M \leq l$. But, there must be a $j \in n$ such that $p_j = l - 1$, since otherwise $Y \in \mathrm{dom}\, S_{l-1}$. Thus, $\mathrm{depth}\, M = l$.

The proof of (f) also proceeds by course of values induction on l, and, again, we consider the case where M has the form $c\, M_0 \cdots M_{n-1}$. If $l = 0$, then $n = 0$, and thus $[\![\lambda x_0 \cdots x_{m-1}.\, M]\!]\, \Delta_0 \cdots \Delta_{m-1} = c_A \in \mathrm{dom}\, S_l$. So, suppose that $l > 0$, so that $n > 0$. Let $p_j < l$ and $Y_j \in A^\alpha_\iota$, for all $j \in n$, be $\mathrm{depth}\, M_j$ and $[\![\lambda x_0 \cdots x_{m-1}.\, M_j]\!]\, \Delta_0 \cdots \Delta_{m-1}$, respectively. Then, by the inductive hypotheses for the p_j's, we have that $Y_j \in \mathrm{dom}\, S_{p_j}$ for all $j \in n$, so that $c_A\, Y_0 \cdots Y_{n-1} \in \mathrm{dom}\, S_l$. But $[\![\lambda x_0 \cdots x_{m-1}.\, M]\!]\, \Delta_0 \cdots \Delta_{m-1} = c_A\, Y_0 \cdots Y_{n-1}$, by Lemma 3.7 (i), and thus we are done.

From (a) and (b) and the fact that there are only finitely many α-tuples over A_ι, we can conclude that there is a largest l such that S_l is defined.

Suppose $\langle X, M\rangle \in S_l$ for some M, so that either $l = 0$ or $X \notin \mathrm{dom}\, S_{l-1}$ (otherwise, S_l would be undefined). Then, the algorithm terminates with a stage of l and the closed term $\lambda x_0 \cdots x_{m-1}.M$, and $\mathrm{depth}\, M = l$ follows by (c). Let $\mathcal{B}$ be a $\Lambda(C)$-model satisfying the specified conditions. Then, $[\![\lambda x_0 \cdots x_{m-1}.\, M]\!]\, \Delta_0 \cdots \Delta_{m-1} = X$ by (e). Furthermore, if $N \in \Gamma(C)_\iota$ is such

Figure 1: Lambda definability problems

$$
\begin{array}{rcl}
\mathit{problem} & \rightarrow & \mathit{iota\_sect}\ \mathit{funs\_sect}\ \mathit{cons\_sect}\ \mathit{tests\_sect} \\
\mathit{iota\_sect} & \rightarrow & \texttt{iota}\ \mathit{Elem}\ \{\ \mathit{Elem}\ \} \\
\mathit{funs\_sect} & \rightarrow & \texttt{functions}\ \{\ \mathit{fun}\ \} \\
\mathit{fun} & \rightarrow & \mathit{Fun}\ \mathit{clause}\ \{\ \mathit{clause}\ \} \\
\mathit{clause} & \rightarrow & \mathit{pat}\ \{\ \mathit{pat}\ \}\ \texttt{=}\ \mathit{result} \\
\mathit{pat} & \rightarrow & \mathit{Elem} \mid \mathit{Var} \mid \texttt{\_} \\
\mathit{result} & \rightarrow & \mathit{Elem} \mid \mathit{Var} \\
\mathit{cons\_sect} & \rightarrow & \texttt{constants}\ \{\ \mathit{con}\ \} \\
\mathit{con} & \rightarrow & \mathit{Elem} \mid \mathit{Fun} \\
\mathit{tests\_sect} & \rightarrow & \texttt{tests}\ \mathit{test}\ \{\ \mathit{test}\ \} \\
\mathit{test} & \rightarrow & \{\ \mathit{test\_arg}\ \}\ \texttt{=}\ \mathit{test\_result} \\
\mathit{test\_arg} & \rightarrow & \mathit{Elem} \mid \mathit{Fun} \\
\mathit{test\_result} & \rightarrow & \mathit{Elem}
\end{array}
$$

that $\mathrm{fv}\, N \subseteq \{x_0, \ldots, x_{m-1}\}$ and $[\![\lambda x_0 \cdots x_{m-1}.\, N]\!]\, \Delta_0 \cdots \Delta_{m-1} = X$, then $\mathrm{depth}\, M \leq \mathrm{depth}\, N$, since otherwise (f) would imply that $X \in \mathrm{dom}\, S_{l'}$ for some $l' < l$.

Otherwise, $X \notin \mathrm{dom}\, S_l$, and thus the algorithm terminates with a stage of l and $\mathrm{dom}\, S_l$. Let $\mathcal{B}$ be a $\Lambda(C)$-model satisfying the specified conditions. By (d) and the fact that S_{l+1} is undefined, we have that $\mathrm{dom}\, S_l = L(\Delta)_\iota$. Thus, there is no definable $b \in B$ such that $b\, \Delta_0 \cdots \Delta_{m-1} = X$, by Theorem 3.9. □

Although instances of Algorithm Schema 3.10 always produce terms of minimal depth, they often fail to produce terms of minimal size. In fact, it is not hard to find an example of a pair of terms with identical depth and meaning, where the first term is produced by a schema instance and the second has strictly smaller size than the first (see the lambda definability problem `size.lam` that is included with *lambda*'s distribution).

4 Implementation

In this section, we describe an implementation, *lambda*, of an instance of Algorithm Schema 3.10, and give several examples of its use. *Lambda* doesn't carry out the algorithm's steps itself. Instead, it takes in a lambda definability problem, representing the algorithm's input data, and generates a C program that solves this problem, producing the algorithm's output.

The grammar in Figure 1 describes the syntax of *lambda definability problems*. In this grammar, curly brackets are used to denote repetition (zero or more occurrences of the phrases they surround). An element name, *Elem*, consists of a single upper case letter or digit. A function name, *Fun*, consists of an upper case letter, followed by one or more letters or digits. A variable name, *Var*, consists of a lower case letter, followed by zero or more lower case letters or digits. As usual, white space characters and comments (which begin with # and continue until end of line) separate tokens but are otherwise ignored.

A lambda definability problem has four sections. The *iota section* lists the elements of the set A_ι—the elements that exist at type ι.

The *functions section* defines zero or more first-order functions, using ML-style pattern matching. Each function definition consists of the function's name followed by a sequence of clauses, each of which must have the same number of patterns in its left hand side. A given variable may not appear twice in the left hand side of the same clause, and, if the right hand side of a clause is a variable, then that variable must appear in the left hand side of that clause.

Suppose that the body of a given function definition has the form

$$\begin{array}{ccccc} p_0^0 & \cdots & p_{m-1}^0 & = & r^0 \\ & & & \vdots & \\ p_0^{n-1} & \cdots & p_{m-1}^{n-1} & = & r^{n-1}. \end{array}$$

A clause j *matches* a sequence of argument elements $a_0, \ldots, a_{m-1}$ iff, for all $i \in m$, the pattern p_i^j is the *wildcard* _ or is a variable or is equal to a_i. The function definition must be completely specified in the sense that it has at least one clause that matches any given sequence of arguments. Furthermore, each of its clauses must be non-redundant in the sense that the clause matches some sequence of elements that isn't matched by any preceding clause in the definition. The function defined by the function definition is the element of A_{ι^m} that sends a sequence of arguments $a_0, \ldots, a_{m-1}$ to r^j, if clause j is the first clause that matches the argument sequence and r^j is an element, and sends the argument sequence to a_i, if clause j is the first clause that matches the argument sequence, r^j is a variable and $p_i^j = r^j$.

The *constants section* specifies the family of constants C, and thus the functions c_A for $c \in C$.

Finally, the *tests section* must have the form

$$\begin{array}{ccccc} \Delta_0^0 & \cdots & \Delta_{m-1}^0 & = & X^0 \\ & & & \vdots & \\ \Delta_0^{\alpha-1} & \cdots & \Delta_{m-1}^{\alpha-1} & = & X^{\alpha-1}. \end{array}$$

It implicitly specifies the natural numbers m and α, the types $\sigma_0, \ldots, \sigma_{m-1}$, the argument tuples $\Delta_0 \in A_{\sigma_0}^\alpha, \ldots, \Delta_{m-1} \in A_{\sigma_{m-1}}^\alpha$ and the result tuple $X \in A_\iota^\alpha$. The number of tests, α, is required to be non-zero, since otherwise a method of explicitly specifying the types σ_i would have to be devised.

Lambda is written in ANSI C, with the exception of its lexical analyzer and parser, which are written in *lex* and *yacc* source, respectively. It uses one UNIX System V system call. The C programs that it generates also conform to the ANSI standard; they use several UNIX System V system calls in order to implement checkpointing. The programs generated by *lambda* make no use of dynamic storage allocation (except during their initialization phases).

A program generated by *lambda* codes tuples of elements as integers, and represents the algorithm's state as an array indexed by those codes. An element of this array records (among other things) whether the tuple coded by its index has been found. If it has, the way in which it was constructed from previously produced tuples is also recorded; implicit in this information is a term that computes the tuple from the argument tuples. When a new tuple is found during a given stage of the closure process, its element of the array is updated to record this fact, but new tuples are distinguished from existing tuples until the stage's end. New tuples are produced by n nested for loops over the tuple codes, where n is the greatest number of arguments that any constant or argument tuple expects. When a given new tuple can be formed in multiple ways, the first way found whose implicit term has minimal size is selected.

Figure 2 contains our first example lambda definability problem (in the left column), along with its solution (in the right column). The comment indicates that this problem is contained in the file `por1.lam` that is included as part of *lambda*'s distribution. We think of `B`, `T` and `F` as standing for the elements $\perp$, tt and ff, respectively, of the monotone function model $\mathcal{F}$ of Finitary PCF. The occurrence of `B` in the constants section stands for the constant Ω of Finitary PCF, which is interpreted as $\perp$ in $\mathcal{F}$. The problem is to determine whether parallel or is definable in models of Finitary PCF that consist of $\{\perp, \mathrm{tt}, \mathrm{ff}\}$ at type ι and in which the constants are interpreted in the same way as in $\mathcal{F}$ (it will either be definable in all or no models of this sort).

Applying *lambda* to `por1.lam` generates a C program that carries out the algorithm's closure process, producing the relation listed in the figure. The stage of one indicates that it took only one stage of this process for the relation to stabilize, and it is easy to see that this relation is the one used to show the non-definability of parallel or in the preceding section. (A triple $\langle x, y, z\rangle$ is in the relation iff $x = y = z$ or $x = \perp$ or $y = \perp$.) Note that the result triple $\langle \mathrm{tt}, \mathrm{tt}, \mathrm{ff}\rangle$ is in the complement of the relation.

Figure 3 shows that parallel or remains non-definable when parallel convergence is added to Finitary PCF (the original proof of this result can be found in [Abr90]). This time the C program produced by *lambda* was run in verbose mode, with the consequence that the elements of the resulting relation are labeled with the stages at which they were found. The relation contains two more triples than does the relation of Figure 2: $\langle \mathrm{tt}, \mathrm{tt}, \perp\rangle$ (found at stage 2) and $\langle \mathrm{ff}, \mathrm{ff}, \perp\rangle$ (found at stage 3).

Figure 4 shows how a non-definability result from Proposition 4.4.2 of [Cur93] can be proved using logical relations. The resulting relation consists of those triples $\langle x, y, z\rangle$ such that $x = y = z$ or one of x, y or z is $\perp$. Oddly, it can be formed by adding two triples to the relation of Figure 3.

Figure 5 shows how the Berry-Plotkin function (cf., Exercise 4.1.18.2 of [Cur93]) can be used to separate Curien's A_1, A_2 and A_3. This time, the program produced by *lambda* was run in both ordinary (middle column) and verbose (right column) modes. The output indicates that the term

$$H = \lambda x_0.\, \mathrm{BP}\,(x_0\,\Omega\,\mathrm{tt}\,\mathrm{ff})\,(x_0\,\mathrm{tt}\,\mathrm{ff}\,\Omega)\,(x_0\,\mathrm{ff}\,\Omega\,\mathrm{tt})$$

(the $\perp$'s have been replaced by Ω's) was found after two stages of the closure process. The verbose version of the program's output shows that the result triple $\langle \mathrm{tt}, \mathrm{ff}, \mathrm{ff}\rangle$ became paired with the body of H at stage 2 of the closure process since

$$\langle \mathrm{tt}, \mathrm{ff}, \mathrm{ff}\rangle = \mathrm{BP}\,\langle \perp, \mathrm{tt}, \mathrm{ff}\rangle\,\langle \mathrm{tt}, \mathrm{ff}, \perp\rangle\,\langle \mathrm{ff}, \perp, \mathrm{tt}\rangle$$

and the triples $\langle \perp, \mathrm{tt}, \mathrm{ff}\rangle$, $\langle \mathrm{tt}, \mathrm{ff}, \perp\rangle$ and $\langle \mathrm{ff}, \perp, \mathrm{tt}\rangle$ were paired with the terms $x_0\,\Omega\,\mathrm{tt}\,\mathrm{ff}$, $x_0\,\mathrm{tt}\,\mathrm{ff}\,\Omega$ and $x_0\,\mathrm{ff}\,\Omega\,\mathrm{tt}$, respectively, at stage 1. Similarly, the triple $\langle \perp, \mathrm{tt}, \mathrm{ff}\rangle$ is paired with the term $x_0\,\Omega\,\mathrm{tt}\,\mathrm{ff}$ at stage 1 since

$$\langle \perp, \mathrm{tt}, \mathrm{ff}\rangle = \langle A_1, A_2, A_3\rangle\,\langle \perp, \perp, \perp\rangle\,\langle \mathrm{tt}, \mathrm{tt}, \mathrm{tt}\rangle\,\langle \mathrm{ff}, \mathrm{ff}, \mathrm{ff}\rangle$$

and the constantly $\perp$, tt and ff triples were paired with the terms Ω, tt and ff at stage 0.

As a final example, we consider the problem of determining whether there is a definable element of type $\iota^3 \to \iota$ of the monotone function model $\mathcal{F}$ of Finitary PCF that sends an argument x to tt,

Figure 2: Non-definability of parallel or

```
# por1.lam

iota

  B T F

functions

  If B _ _ = B
     T x _ = x
     F _ y = y

constants

  B T F If

tests

  T B = T
  B T = T
  F F = F
```

```
por1

Stage: 1

Relation (17 elements):

<B B B>
<B B T>
<B B F>
<B T B>
<B T T>
<B T F>
<B F B>
<B F T>
<B F F>
<T B B>
<T B T>
<T B F>
<T T T>
<F B B>
<F B T>
<F B F>
<F F F>

Relation complement (10 elements):

<T T B>
<T T F>
<T F B>
<T F T>
<T F F>
<F T B>
<F T T>
<F T F>
<F F B>
<F F T>
```

Figure 3: Parallel or is not definable using parallel convergence

```
# por2.lam

iota

  B T F

functions

  If B _ _ = B
     T x _ = x
     F _ y = y

  PConv B B = B
          _ _ = T

constants

  B T F If PConv

tests

  T B = T
  B T = T
  F F = F
```

```
por2

Stage: 3

Relation (19 elements):

<B B B>  0
<B B T>  1
<B B F>  1
<B T B>  1
<B T T>  1
<B T F>  0
<B F B>  1
<B F T>  1
<B F F>  1
<T B B>  1
<T B T>  1
<T B F>  0
<T T B>  2
<T T T>  0
<F B B>  1
<F B T>  1
<F B F>  1
<F F B>  3
<F F F>  0

Relation complement (8 elements):

<T T F>
<T F B>
<T F T>
<T F F>
<F T B>
<F T T>
<F T F>
<F F T>
```

Figure 4: The impossibility of separating Curien's A_1, A_2 and A_3.

```
# curien1.lam

iota

  B T F

functions

  If B _ _ = B
     T x _ = x
     F _ y = y

  A1 T F _ = T
     F _ T = F
     _ _ _ = B

  A2 _ T F = T
     T F _ = F
     _ _ _ = B

  A3 F _ T = T
     _ T F = F
     _ _ _ = B

constants

  B T F If

tests

  A1 = T
  A2 = F
  A3 = F
```

```
curien1

Stage: 2

Relation (21 elements):

<B B B>
<B B T>
<B B F>
<B T B>
<B T T>
<B T F>
<B F B>
<B F T>
<B F F>
<T B B>
<T B T>
<T B F>
<T T B>
<T T T>
<T F B>
<F B B>
<F B T>
<F B F>
<F T B>
<F F B>
<F F F>

Relation complement (6 elements):

<T T F>
<T F T>
<T F F>
<F T T>
<F T F>
<F F T>
```

Figure 5: The Berry-Plotkin function can be used to separate A_1, A_2 and A_3.

```
# curien3.lam

iota

  B T F

functions

  If B _ _ = B
     T x _ = x
     F _ y = y

  A1 T F _ = T
     F _ T = F
     _ _ _ = B

  A2 _ T F = T
     T F _ = F
     _ _ _ = B

  A3 F _ T = T
     _ T F = F
     _ _ _ = B

  BP T F _ = F
     _ T F = T
     F _ T = F
     _ _ _ = B

constants

  B T F If BP

tests

  A1 = T
  A2 = F
  A3 = F
```

```
curien3

Stage: 2

Term:

lambda x0.
    BP
        x0
            B
            T
            F
        x0
            T
            F
            B
        x0
            F
            B
            T
```

```
curien3

Stage: 2

Term:

lambda x0.
    BP    <T F F>
        x0    <B T F>
            B     <B B B>
            T     <T T T>
            F     <F F F>
        x0    <T F B>
            T     <T T T>
            F     <F F F>
            B     <B B B>
        x0    <F B T>
            F     <F F F>
            B     <B B B>
            T     <T T T>
```

if $x \sqsupseteq A_i$ for some i, and sends x to $\perp$, otherwise. Since there are many elements of F_{ι^3} that don't dominate any of the A_i's, *lambda* can't be used in a purely mechanical way to solve this problem.

One can, however, use *lambda* to solve lambda definability problems that specify that certain hand-picked functions must be sent to $\perp$. A bit of experimentation (see `curien4.lam` and `curien5.lam` in *lambda*'s distribution) lead to the problem of Figure 6, which specifies that parallel or, parallel and, and their "negations" should be sent to $\perp$. Running the program generated from this problem by *lambda* takes a considerable amount of time (about eight hours of cpu time on a Sun 690MP) and produces the term $\lambda x_0.\, G$, where

$$\begin{aligned}
G &= x_0\, L\, M\, N \\
L &= \mathrm{If}\, Z\, (\mathrm{If}\, Y\, \Omega\, \mathrm{ff})\, (\mathrm{If}\, X\, \mathrm{tt}\, \Omega) \\
M &= \mathrm{If}\, X\, (\mathrm{If}\, Z\, \Omega\, \mathrm{ff})\, (\mathrm{If}\, Y\, \mathrm{tt}\, \Omega) \\
N &= \mathrm{If}\, Y\, (\mathrm{If}\, X\, \Omega\, \mathrm{ff})\, (\mathrm{If}\, Z\, \mathrm{tt}\, \Omega) \\
X &= x_0\, \mathrm{tt}\, \mathrm{ff}\, \Omega \\
Y &= x_0\, \Omega\, \mathrm{tt}\, \mathrm{ff} \\
Z &= x_0\, \mathrm{ff}\, \Omega\, \mathrm{tt}.
\end{aligned}$$

By considering the possible values of X, Y and Z, it is straightforward to show that G produces tt iff x_0 dominates one of the A_i's or is the constantly tt function. Furthermore, the term $\lambda x_0.\, H$, where

$$H = x_0\, M\, N\, L,$$

is produced as the solution of the variation of this problem (called `curien7.lam` in *lambda*'s distribution) that specifies that the A_i's should be sent to ff rather than to tt, and H produces ff iff x_0 dominates one of the A_i's or is the constantly ff function. Thus, it is easy to see that the term

$$Q = \lambda x_0.\, \mathrm{If}\, G\, (\mathrm{If}\, H\, \Omega\, \mathrm{tt})\, \Omega$$

solves the problem of sending an argument to tt, when it dominates one of the A_i's, and sending the argument to $\perp$, otherwise.

Interestingly, I wasn't able to generate such a term as the solution to a single lambda definability problem. One obstacle to my doing so was the necessity of employing at most seven tests, since it would take weeks rather than hours to solve a problem with eight tests. In any event, there is no chance of producing Q itself in such a way, since its body has depth six and there is another known solution to the problem whose body has depth five.

Acknowledgments

It is a pleasure to acknowledge many fruitful discussions with Achim Jung. Conversations with Antonio Bucciarelli, Shai Geva, Alan Jeffrey, Ralph Loader and Edmund Robinson were also helpful.

References

[Abr90] S. Abramsky. The lazy lambda calculus. In D. L. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1990.

Figure 6: Synthesis of a term sending the A's to tt and parallel or, parallel and, and their negations to $\perp$.

```
# curien6.lam

iota

  B T F

functions

  If B _ _ = B
     T x _ = x
     F _ y = y

  A1 T F _ = T
     F _ T = F
     _ _ _ = B

  A2 _ T F = T
     T F _ = F
     _ _ _ = B

  A3 F _ T = T
     _ T F = F
     _ _ _ = B

  POr F F F = F
      T _ _ = T
      _ T _ = T
      _ _ T = T
      _ _ _ = B

  NPOr F F F = T
       T _ _ = F
       _ T _ = F
       _ _ T = F
       _ _ _ = B

  PAnd T T T = T
       F _ _ = F
       _ F _ = F
       _ _ F = F
       _ _ _ = B

  NPAnd T T T = F
        F _ _ = T
        _ F _ = T
        _ _ F = T
        _ _ _ = B

constants

  B T F If

tests

  A1    = T
  A2    = T
  A3    = T
  POr   = B
  NPOr  = B
  PAnd  = B
  NPAnd = B
```

[Cur93] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Progress in Theoretical Computer Science. Birkhäuser, 1993.

[JS93] A. Jung and A. Stoughton. Studying the fully abstract model of PCF within its continuous function model. In M. Bezem and J. F. Groote, editors, *International Conference on Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 230–244. Springer-Verlag, 1993.

[JT93] A. Jung and J. Tiuryn. A new characterization of lambda definability. In M. Bezem and J. F. Groote, editors, *International Conference on Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 245–257. Springer-Verlag, 1993.

[Loa94] R. Loader. The undecidability of λ-definability. In M. Zeleny, editor, *The Church Festschrift.* CSLI/University of Chicago Press, 1994.

[Mit90] J. C. Mitchell. Type systems for programming languages. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 8, pages 367–458. Elsevier Science Publishers, 1990.

[Plo77] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–256, 1977.

[Plo80] G. D. Plotkin. Lambda-definability in the full type hierarchy. In J. Seldin and J. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 363–374. Academic Press, 1980.

[Sie92] K. Sieber. Reasoning about sequential functions via logical relations. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science*, volume 177 of *LMS Lecture Note Series*, pages 258–269. Cambridge University Press, 1992.