An alternate proof of Statman's finite completeness theorem

B. Srivathsan, Igor Walukiewicz

LaBRI, Université de Bordeaux, 351 cours de la libération, 33400 Talence, France

Abstract

Statman's finite completeness theorem says that for every pair of non-equivalent terms of simply-typed lamda-calculus there is a model that separates them. A direct method of constructing this model is provided using a simple induction on the Böhm tree of the term.

Keywords: Simply typed lambda calculus, formal semantics, theory of computation

1. Introduction

Statman's finite completeness theorem [5, 6] shows that standard models are strong enough to separate terms, upto $\beta\eta$ reductions. It states that given a simply typed lambda term M, there exists a finite *standard model* [1] such that for every term N that is not $\beta\eta$-equivalent to M there is a variable assignement separating the two terms: making their values in the model different. At the time of publication of this work, a crucial corollary of this theorem, again proved in [5, 6], was that the λ-definability conjecture implies the higher order matching conjecture [5, 6, 7]. However, λ-definability was shown to be undecidable later by Loader in [2].

The first proof of this theorem appeared in [5]. It was explained in more detail in [6] since the previous proof was considered "not accessible to readers not familiar with this subject" [6]. The proof proceeds by definining a suitable syntactic equivalence over the lambda terms. The required model is then the set of lambda terms quotient with respect to this equivalence.

Salvati in [4] proves that *singleton sets*, that is sets of the form $\{N | N =_{\beta\eta} M\}$ can be characterized by suitable *intersection types*. In another paper [3], Salvati gives a notion of recognizability of languages of lambda terms based on these intersection types. Additionally, another definition of recognizability is also provided using finite standard models in the same work, and it is shown to be equivalent to the recognizability in terms of intersection types. This provides an alternate proof to Statman's finite completeness theorem.

In this paper, we give yet another proof of this theorem. Our proof carries a semantic flavour, constructing the required model for a term M step-by-step,

by performing an induction on the Böhm tree of the η-long β normal form of M. The Böhm trees are the only syntactic tools used. This proof is very direct, especially in comparison to the existing proofs mentioned above. The proof also gives a slightly stronger result: for every term M there is a model and a valuation such that if N evaluates to the same value as M then $M =_{\beta\eta} N$.

In Section 2, we give the necessary preliminaries. In Section 3, we define the notion of an *extended model*, and explain the relation between the elements of the initial model and the extended model. Section 4 contains our proof of the finite completeness theorem.

2. Simply typed λ-calculus

The set of *types* $\mathcal{T}$ is constructed from a unique *basic type* 0 using a binary operator $\to$. Thus 0 is the unique basic type, and if α, β are types, then $\alpha \to \beta$ is also a type. The order of a type is defined by: $order(0) = 1$, and $order(\alpha \to \beta) = max(1 + order(\alpha), order(\beta))$.

The set of *simply typed λ terms* is defined inductively as follows. For each type α, there is a countable set of variables $x^\alpha, y^\alpha, \ldots$ which are also terms of type α. If M is a term of type β and x is a variable of type α, then $\lambda x^\alpha.M$ is a term of type $\alpha \to \beta$. Such a term is called a λ-*abstraction*. If M is a term of type $\alpha \to \beta$ and N is a term of type α then MN is a term of type β. Terms of this kind are called *applications*.

A *standard finite model* $\mathcal{D}$ is a family of finite sets $(D_\alpha)_{\alpha\in\mathcal{T}}$ indexed by the set of types. $\mathcal{D}$ is determined by D_0 which is a finite set of elements of type 0. For types α,β, the set $D_{\alpha\to\beta}$ is the set of functions from D_α to D_β.

A *variable assignment* is a function assigning to every variable x^α an element of D_α. If d is an element of D_α and x^α is a variable of type α, $v[d/x^\alpha]$ denotes the variable assignment which assigns d to x^α and is identical to v otherwise.

The *interpretation* of a simply typed λ-term M in the model $\mathcal{D}$ and variable assignment v is defined inductively:

- $[\![x^\alpha]\!]^v_{\mathcal{D}} = v(x^\alpha)$
- $[\![MN]\!]^v_{\mathcal{D}} = [\![M]\!]^v_{\mathcal{D}}[\![N]\!]^v_{\mathcal{D}}$
- $[\![\lambda x^\alpha.M]\!]^v_{\mathcal{D}}$ is a function mapping $d \in D_\alpha$ to $[\![M]\!]^{v[d/x^\alpha]}_{\mathcal{D}}$

We recall the two types of reduction over simply typed λ terms.

β-reduction $(\lambda x.M)N \to_\beta M[N/x]$

η-reduction $(\lambda x.Mx) \to_\eta M$, provided x is not free in M.

A lambda term in *long normal form* is of the shape $\lambda\vec{x}.zM_1 \ldots M_k$ where $M_1, \ldots, M_k$ are in long normal form, z is a variable, the term $zM_1 \ldots M_k$ is of type 0 and the sequence $\lambda\vec{x}$ might be empty.

For a lambda term M in long normal form, its Böhm tree, $BT(M)$ is defined inductively as follows. If $M = \lambda\vec{x}.zM_1 \ldots M_k$, with z being a variable, then the root of $BT(M)$ is labeled $\lambda\vec{x}.z$ and it has $BT(M_1)$ to $BT(M_k)$ as its children.

M is said to be *uniquely determined* in a model $\mathcal{D}$ with a variable assignment v if for all lambda terms N, $[\![N]\!]^v_{\mathcal{D}} = [\![M]\!]^v_{\mathcal{D}}$ iff $N =_{\beta\eta} M$.

In the following sections, we prove Statman's finite completeness theorem in a slighlty stronger form:

Theorem 1 *For every λ-term M, there exists a finite model $\mathcal{D}$ and a variable assignment v such that M is uniquely determined in $\mathcal{D}$ and v.*

To prove this theorem, we consider a lambda term in long normal form. We construct a model in which all its subterms are uniquely determined. An additional element is added and the interpretations then altered to make the lambda term interpret uniquely to this newly added element.

3. Extended model

Consider a lambda term M of type 0. Let $\mathcal{D}$ be a standard finite model and v a variable assignment, so that $[\![M]\!]^v_{\mathcal{D}} = e$, with $e \in D_0$. In general, there exist many lambda terms that interpret to e. Our objective is to add a new element to D_0 and make M interpret to this new element. In addition, the other lambda terms of type 0 should interpret as before. This would ensure that M interprets uniquely to this new element. Intuitively, the other lambda terms should not "notice" a difference between e and this new element. We call this new element e_{clone}. Given a model $\mathcal{D} = (D_\alpha)_{\alpha\in\mathcal{T}}$ and an element $e \in D_0$, the *extended model* $\mathcal{D}^e = (D^e_\alpha)_{\alpha\in\mathcal{T}}$ is the model determined by $D^e_0 = D_0 \uplus \{e_{clone}\}$. As a consequence of adding this extra element, many new higher order functions are generated. Hence we force the λ-terms to interpret to those functions that behave identically on e_{clone} and on e. In the subsequent sections, we study this new *extended model* and furnish a variable assignment so that M gets uniquely interpreted to e_{clone}.

3.1. Relating the models

Consider the function $f \in D_{0\to 0}$ shown in Figure 1. The same figure shows some functions in the extended model $\mathcal{D}^e$. The function f'_1 acts the same way as f on all the common elements. However, $f'_1(e_{clone})$ is not equal to $f'_1(e)$ which is undesirable. Hence we would like to ignore such a function. The function f'_2 on the other hand acts the same way as f on all the common elements and in addition $f'_2(e_{clone})$ is equal to $f'_2(e)$. We consider f'_2 as the *representative* of f in $\mathcal{D}^e$. An interesting case is given by f'_3 that instead of mapping the element to e maps it to e_{clone}. By the intuition that e_{clone} is *equivalent* to e, we wish to say that f'_3 is *equivalent* to f'_2.

We define two notions to relate the elements of the extended model $\mathcal{D}^e$ to elements of the original model $\mathcal{D}$:

- an injection function $\mathbf{in}_\alpha\alpha \to D^e_\alpha$ that for every element $f \in D_\alpha$ gives its *representative* $f' \in D^e_\alpha$,

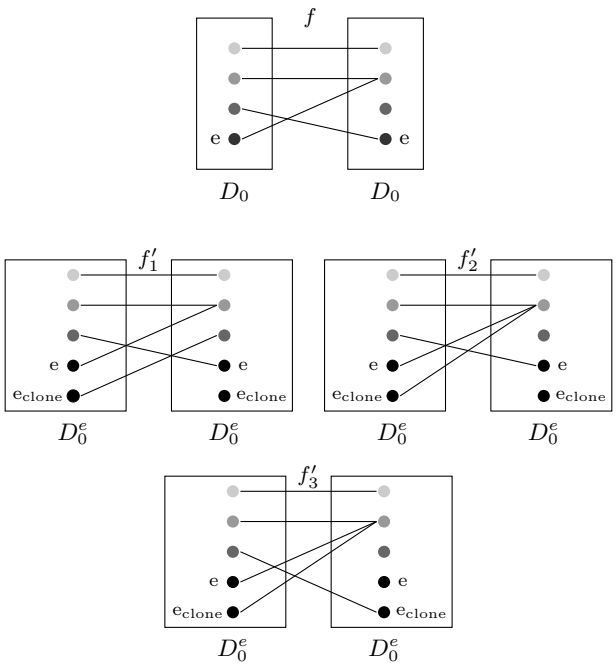

Figure 1: Higher order functions in the extended model $\mathcal{D}^e$

- an equivalence relation $\leftrightarrow_\alpha$ over D^e_α that groups e and e_{clone} at type 0 and propagates this basic equivalence to higher order functions.

In general, we would like to visualize each set D^e_α as shown in Figure 2.

Before formally defining these notions we designate a *null element* for every type.

Definition 2 The *null element* Δ_0 is any arbitrary element of D^e_0 different from e_{clone}. For a type $\alpha \to \beta$, element $\Delta_{\alpha\to\beta}$ is the constant function mapping every element to Δ_β.

The definitions of $\mathbf{in}_\alpha$ and $\leftrightarrow_\alpha$ are mutually dependent. For an element d' in D^e_α, let $[d']$ denote the equivalence class of d' with respect to $\leftrightarrow_\alpha$. For a higher order type $\alpha \to \beta$ and for a function $f \in D_{\alpha\to\beta}$, $\mathbf{in}_{\alpha\to\beta}(f)$ maps every element d' in $[\mathbf{in}_\alpha(d)]$ to $\mathbf{in}_\beta(f(d))$. We say that a function $f' \in D^e_{\alpha\to\beta}$ *simulates* a function $f \in D_{\alpha\to\beta}$, written as $sim(f', f)$ if f' maps every element in an equivalence class $[\mathbf{in}_\alpha(d)]$ to an element in the equivalence class $[\mathbf{in}_\beta(f(d))]$. These notions are pictorially represented in Figure 3. The equivalence relation $\leftrightarrow_{\alpha\to\beta}$ groups functions of $\mathcal{D}^e$ that *simulate* the same function of $\mathcal{D}$. The formal definitions follow.

Definition 3 $\mathbf{in}_\alpha$, sim_α, $\leftrightarrow_\alpha$

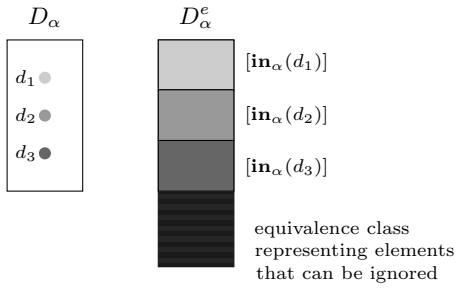

Figure 2: Visualizing a set in the extended model

- $\mathbf{in}_0$, sim_0, $\leftrightarrow_0$
  - $\mathbf{in}_0 : D_0 \to D^e_0$ is the identity.
  - $sim_0(d, d)$ for every element $d \in D_0$.
  - $\leftrightarrow_0$ is the smallest equivalence containing $e \leftrightarrow_0 e_{clone}$.
- $\mathbf{in}_{\alpha\to\beta}$

  For an element $f \in D_{\alpha\to\beta}$, $\mathbf{in}_{\alpha\to\beta}(f)$ is a function $f' \in D^e_{\alpha\to\beta}$ such that for all elements $d' \in D^e_\alpha$,

$$f'(d') = \begin{cases} \mathbf{in}_\beta(f(d)) & \text{if } d' \in [\mathbf{in}_\alpha(d)] \\ \Delta_\beta & \text{otherwise} \end{cases}$$

- $sim_{\alpha\to\beta}$

  For $f \in D_{\alpha\to\beta}$, $f' \in D^e_{\alpha\to\beta}$, we say f' *simulates* f, written as $sim(f', f)$, if for all $d \in D_\alpha$, for all $d' \in [\mathbf{in}_\alpha(d)]$: $f'(d') \leftrightarrow_\beta \mathbf{in}_\beta(f(d))$.

- $\leftrightarrow_{\alpha\to\beta}$

  For $f', g' \in D^e_{\alpha\to\beta}$, $f' \leftrightarrow_{\alpha\to\beta} g'$ if for all $h \in D_{\alpha\to\beta}$, $sim(f', h) \Leftrightarrow sim(g', h)$.

Remark 4 Subsequently, we drop the type subscript in $\mathbf{in}_\alpha$, sim_α and $\leftrightarrow_\alpha$ since it is the same as the type of the elements associated.

Lemma 5 For every $d \in \mathcal{D}$, $\mathbf{in}(d)$ *simulates* d.

Proof
The lemma is direct for type 0. For a higher order function $f \in D_{\alpha\to\beta}$, it follows from the definitions. □

Lemma 6 For $d, d_1, d_2 \in D_\alpha$ and $d' \in D^e_\alpha$,

1. $sim(d', d_1)$ and $sim(d', d_2)$ implies $d_1 = d_2$,

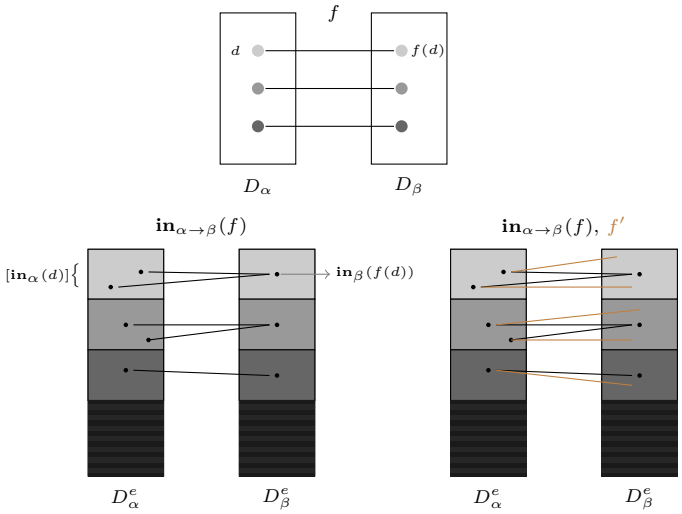

Figure 3: f, $\mathbf{in}_{\alpha\to\beta}(f)$, $sim(f', f)$

2. $sim(d', d) \Leftrightarrow d' \leftrightarrow \mathbf{in}(d)$,
3. $d_1 \neq d_2 \Rightarrow \mathbf{in}(d_1) \nleftrightarrow \mathbf{in}(d_1)$.

Proof
The proof proceeds by induction on the types. The lemma is clear for type 0. We prove the lemma for a higher order type $\alpha \to \beta$. Consider $f, f_1, f_2 \in D_{\alpha\to\beta}$ and $f' \in D^e_{\alpha\to\beta}$.

1. Suppose $sim(f', f_1)$ and $sim(f', f_2)$. Take $d \in D_\alpha$ and $d' \in [\mathbf{in}(d)]$. By definition of sim, $f'(d') \leftrightarrow \mathbf{in}(f_1(d))$ and $f'(d') \leftrightarrow \mathbf{in}(f_2(d))$. Hence $\mathbf{in}(f_1(d)) \leftrightarrow \mathbf{in}(f_2(d))$ and by 3), $f_1(d) = f_2(d)$. Since d is arbitrary, $f_1 = f_2$.
2. Suppose $sim(f', f)$. By 1) if $sim(f', h)$ then $h = f$. Since from Lemma 5, $sim(\mathbf{in}(f), f)$, the same holds for $\mathbf{in}(f)$. Therefore, for all h, $sim(f', h) \Leftrightarrow sim(in(f), h)$ and hence by definition of $\leftrightarrow$, $f' \leftrightarrow \mathbf{in}(f)$.
   Suppose $f' \leftrightarrow \mathbf{in}(f)$. By Lemma 5, $sim(\mathbf{in}(f), f)$ and by definition of sim, $sim(f', f)$.
3. Suppose $f_1 \neq f_2$. From Lemma 5, $sim(\mathbf{in}(f_1), f_1)$. Hence by 1), not $sim(\mathbf{in}(f_1), f_2)$. But since $sim(\mathbf{in}(f_2), f_2)$, we get $\mathbf{in}(f_1) \nleftrightarrow \mathbf{in}(f_2)$.

□

3.2. Interpreting the lambda terms in the extended model

To interpret the lambda terms in $\mathcal{D}^e$, we need to define the variable assignment v^e that interprets the variables. We intend to pick one from a set of variable assignments that *simulate* v.

Definition 7 A variable assignment v' on $\mathcal{D}^e$ *simulates* a variable assignment v on $\mathcal{D}$ if for all variables x: $sim(v'(x), v(x))$.

Lemma 8 If v' *simulates* v then for every lambda term M:

$$sim([\![M]\!]^{v'}_{\mathcal{D}^e}, [\![M]\!]^{v}_{\mathcal{D}})$$

Proof
We proceed by induction on the structure of the lambda term.

1. For variables, the lemma follows from the hypothesis.
2. Consider an application MN, with M of type $\alpha \to \beta$ and N of type α. By induction, $sim([\![N]\!]^{v'}_{\mathcal{D}^e}, [\![N]\!]^{v}_{\mathcal{D}})$ and hence from 2) of Lemma 6, $[\![N]\!]^{v'}_{\mathcal{D}^e} \leftrightarrow \mathbf{in}([\![N]\!]^{v}_{\mathcal{D}})$. Also by induction, $sim([\![M]\!]^{v'}_{\mathcal{D}^e}, [\![M]\!]^{v}_{\mathcal{D}})$ and hence from definition, $[\![M]\!]^{v'}_{\mathcal{D}^e}([\![N]\!]^{v'}_{\mathcal{D}^e}) \leftrightarrow \mathbf{in}([\![M]\!]^{v}_{\mathcal{D}}([\![N]\!]^{v}_{\mathcal{D}}))$. Therefore by 2) of Lemma 6, $sim([\![MN]\!]^{v'}_{\mathcal{D}^e}, [\![MN]\!]^{v}_{\mathcal{D}})$.
3. Consider a lambda abstraction $\lambda x^\alpha.M$. Take $d \in D_\alpha$ and $d' \in [\mathbf{in}(d)]$. Since $sim(v', v)$, we have $sim(v'[d'/x^\alpha], v[d/x^\alpha])$ and hence by induction $sim([\![M]\!]^{v'[d'/x^\alpha]}_{\mathcal{D}^e}, [\![M]\!]^{v[d/x^\alpha]}_{\mathcal{D}})$. From 2) of Lemma 6, $[\![M]\!]^{v'[d'/x^\alpha]}_{\mathcal{D}^e} \leftrightarrow [\![M]\!]^{v[d/x^\alpha]}_{\mathcal{D}}$. This is true for all $d \in D_\alpha$. Hence, by definition $sim([\![\lambda x^\alpha.M]\!]^{v'}_{\mathcal{D}^e}, [\![\lambda x^\alpha.M]\!]^{v}_{\mathcal{D}})$.

□

Corollary 9 If v' *simulates* v, then every term uniquely determined in $(\mathcal{D}, v)$ is uniquely determined in $(\mathcal{D}^e, v')$.

Proof
Let M be uniquely determined in $(\mathcal{D}, v)$ but not in $(\mathcal{D}^e, v')$. Therefore, there exists $N \neq_{\beta\eta} M$ such that $[\![N]\!]^{v'}_{\mathcal{D}^e} = [\![M]\!]^{v'}_{\mathcal{D}^e}$. From Lemma 8, this would mean that $sim([\![M]\!]^{v'}_{\mathcal{D}^e}, [\![M]\!]^{v}_{\mathcal{D}})$ and $sim([\![M]\!]^{v'}_{\mathcal{D}^e}, [\![N]\!]^{v}_{\mathcal{D}})$. Hence by 1) of Lemma 6, $[\![M]\!]^{v}_{\mathcal{D}} = [\![N]\!]^{v}_{\mathcal{D}}$. A contradiction. □

4. Proof of the theorem

The proof proceeds by an induction on the size of the Böhm tree $BT(M)$ of the lambda term M. Let $BT(M)$ contain m nodes. Consider an ordering $s_1 < \cdots < s_m$ of the nodes of $BT(M)$ that satisfies the condition that if a node s_i is a child of s_j, then $s_i < s_j$. Assume that $\mathcal{D}_k$ is a model and v_k a variable assignment such that all the lambda terms rooted in the nodes s_i with $i \leq k$ are uniquely determined in $(\mathcal{D}_k, v_k)$. We then construct $(\mathcal{D}_{k+1}, v_{k+1})$ where all the lambda terms rooted in the nodes s_i with $i \leq k+1$ are uniquely determined. Consequently M gets uniquely determined in (D_m, v_m).

Base case

The base case refers to $(\mathcal{D}_1, v_1)$ which uniquely determines a leaf of $BT(M)$. A leaf is variable z of type 0. Starting with the trivial model $\mathcal{D}_0$ which has a singleton $\{\perp\}$ in its basic set and the trivial variable assignment v_0, we construct the extended model $\mathcal{D}_0^e$ by adding a new element $\perp_{clone}$ to the atomic set. The new variable assignment v_0^e assigns z to $\perp_{clone}$ and the rest of the variables are maintained with the same interpretation. Clearly, z is uniquely determined in $(\mathcal{D}_0^e, v_0^e)$. Set $\mathcal{D}_1$ as $\mathcal{D}_0^e$ and v_1 as v_0^e.

Induction case

Let the lambda term rooted at s_k be $\lambda\vec{x}.yM_1 \dots M_n$ and let $[\![yM_1 \dots M_n]\!]^{v_k}_{\mathcal{D}_k} = e$. For notational simplicity let $\mathcal{D} = \mathcal{D}_k$ and $v = v_k$. By induction hypothesis, $M_1, \dots, M_n$ are uniquely determined in $(\mathcal{D}, v)$.

Construct the extended model $\mathcal{D}^e$ by adding an element e_{clone} to the basic set D_0 of $\mathcal{D}$. Consider the variable assignment v^e defined below.

- $v^e(x) = \mathbf{in}(v(x))$, if $x \neq y$.
- For the variable y,

$$v^e(y)(d'_1, \dots, d'_n) = \begin{cases} e_{clone} & \text{if } d'_i \in [\mathbf{in}([\![M_i]\!]^v_{\mathcal{D}})], \\ & \text{for } i \in \{1, \dots, n\} \\ \mathbf{in}(v(y))(d'_1, \dots, d'_n) & \text{otherwise} \end{cases}$$

Since $e_{clone} \leftrightarrow e$, v^e *simulates* v. Hence we infer the following.

1. From Lemma 8, for every lambda term N, $[\![N]\!]^{v^e}_{\mathcal{D}^e}$ *simulates* $[\![N]\!]^v_{\mathcal{D}}$, and hence from Lemma 6
$$[\![N]\!]^{v^e}_{\mathcal{D}^e} \leftrightarrow \mathbf{in}([\![N]\!]^v_{\mathcal{D}})$$
2. $[\![yM_1 \dots M_n]\!]^{v^e}_{\mathcal{D}^e} = e_{clone}$.

We now prove that $[\![yM_1 \dots M_n]\!]^{v^e}_{\mathcal{D}^e}$ is uniquely interpreted to e_{clone}. Let $zN_1 \dots N_p$ be a lambda term such that $[\![zN_1 \dots N_p]\!]^{v^e}_{\mathcal{D}^e} = e_{clone}$. If $z \neq y$, then $v^e(z) = \mathbf{in}(v(z))$. However, observe that there does not exist an element $d \in D_0$ such that $\mathbf{in}(d) = e_{clone}$. Also, note that $\Delta_0 \neq e_{clone}$. Hence by definition, $\mathbf{in}(v(z))(d'_1, \dots, d'_p)$ cannot be equal to e_{clone} for any values of $d'_1, \dots, d'_p$ implying $z = y$.

Since $z = y$, p equals n. We show that $N_i = M_i$ for all i. Now, if $[\![N_i]\!]^{v^e}_{\mathcal{D}_e} \notin [\mathbf{in}([\![M_i]\!]^v_{\mathcal{D}})]$ for some i, by the same reasoning as above, $[\![zN_1 \dots N_p]\!]^{v^e}_{\mathcal{D}^e}$ cannot be equal to e_{clone}. Therefore, $[\![N_i]\!]^{v^e}_{\mathcal{D}_e} \leftrightarrow [\![M_i]\!]^v_{\mathcal{D}}$ for all i. In addition, from Lemma 8, we know that $[\![N_i]\!]^{v^e}_{\mathcal{D}_e} \leftrightarrow [\![N_i]\!]^v_{\mathcal{D}}$ too. Hence from the third part of Lemma 6, $[\![N_i]\!]^v_{\mathcal{D}} = [\![M_i]\!]^v_{\mathcal{D}}$. From the assumption that each M_i is uniquely determined in $(\mathcal{D}, v)$, one can deduce that $N_i = M_i$ for $i \in \{1, \dots, n\}$. We hence infer that $yM_1 \dots M_k$ is uniquely determined in $(\mathcal{D}^e, v^e)$.

Note that this implies $\lambda\vec{x}.yM_1 \dots M_k$ is uniquely determined too in $(\mathcal{D}^e, v^e)$ since, for another lambda term $\lambda\vec{x}.N$, if $[\![\lambda\vec{x}.N]\!]^{v^e}_{\mathcal{D}^e} = [\![\lambda\vec{x}.yM_1 \dots M_k]\!]^{v^e}_{\mathcal{D}^e}$, then N with $\vec{x}$ substituted by values from v^e and $yM_1 \dots M_k$ with $\vec{x}$ substituted by values from v^e interpret to the same element of $\mathcal{D}^e$, contradicting the fact that $yM_1 \dots M_k$ is uniquely determined with the variable assignment v^e.

Set $\mathcal{D}_{k+1} = \mathcal{D}^e$ and $v_{k+1} = v^e$. Therefore, from the above argument and from Corollary 9, the lambda terms rooted at nodes s_i with $i \leq k+1$ are uniquely determined in $(\mathcal{D}_{k+1}, v_{k+1})$, thus proving the inductive step.

[1] L. Henkin. Completeness in the theory of types. *Journal of symbolic logic*, 15(2):81–91, 1950.

[2] R. Loader. The undecidability of-definability. *Logic, meaning, and computation: essays in memory of Alonzo Church*, page 331, 2001.

[3] S. Salvati. Recognizability in the simply typed lambda-calculus. *Logic, Language, Information and Computation*, pages 48–60, 2009.

[4] S. Salvati. On the membership problem for non-linear Abstract Categorial Grammars. *Journal of Logic, Language and Information*, 19(2):163–183, 2010.

[5] R. Statman. Completeness, invariance and λ-definability. *Journal of Symbolic Logic*, 47(1):17–26, 1982.

[6] R. Statman and G. Dowek. On Statman's Finite Completeness Theorem. *Carnegie Mellon University, School of Computer Science*, 1992.

[7] DA Wolfram. *The clausal theory of types*. PhD thesis, 1989.