An Algorithm for the Minimal Model

Ralph Loader, Merton College, Oxford.

July, 1997. Version 1.4, dated 2001/11/27 06:02:41 UTC.

Abstract

Padovani [?] gave algorithms that (a) calculate a presentation of the minimal model of the simply typed lambda calculus, and (b) decides the equational theory of that model. Loader [?] generalised that work to unary PCF. Schmidt-Schauß [?] gave much simpler algorithms and proofs (based on strictness analyses) for unary PCF. Here, we present Schmidt-Schauß's method restricted to the minimal model.

1 Introduction

We consider the simply typed lambda calculus with a single ground type o and two constants $\top$ and $\bot$, both of ground type. We order $\top$ and $\bot$ by setting $\bot \leqslant \top$, $\top \nleqslant \bot$. This is extended to all (closed) terms of ground type by comparing normal forms. There are two natural ways of extending this to higher types. The pointwise, applicative ordering is given by

$$f \leqslant g \qquad \text{if and only if} \qquad f\,a \leqslant g\,a\ (\forall a).$$

Alternative we can extend as a logical relation:

$$f \leqslant g \qquad \text{if and only if} \qquad f\,a \leqslant g\,b\ (\forall\, a, b \text{ s.t. } a \leqslant b).$$

(In both cases, f and g are two terms of the same function type, a and b range over terms of appropriate type.)

It is well known that these two extensions in fact coincide. This can be demonstrated either by using the logical relations lemma to show that the logical relation is reflexive, or by proving a context lemma for the applicative ordering. Note that the applicative ordering is clearly a pre-order on the terms of each type.

The equivalence $\equiv$ is defined to be $(\leqslant) \cap (\geqslant)$. There are two equivalence classes at ground type, namely those of $\top$ and $\bot$. Note that if there are m equivalence classes of type A and n of type B, then there are at most n^m classes of type $A \to B$. In particular, by induction, there are only finitely many equivalence classes of each type.

We shall derive a reasonably simple algorithm that given a type A, calculates a finite list of terms of type A, containing at least one representative of each equivalence class of type A. We shall call such an algorithm a **listing algorithm**. (In actual fact, the lists computed by our algorithm contain exactly one representative of each class.)

Lemma 1 *Given a listing algorithm, we can decide* $\equiv$ *and* $\leqslant$.

Proof: Let r and s be two terms of type $A = A_1 \to \cdots \to A_n \to o$. We can decide $r \equiv s$ and $r \leqslant s$ as follows: use the listing algorithm to compute complete lists of representatives at each type $A_1, \ldots, A_n$. Now compare r and s pointwise at each combination of parameters from these complete lists. $\square$

We define some notation and combinators that will be useful later:

- We shall abbreviate the type $A_1 \to \cdots \to A_m \to o$ to $[A_1 \ldots A_m]$.
- Given a type $A = [A_1 \ldots A_m]$, define $K^A = (\lambda u\, x_1 \ldots x_m)\, u : o \to A$.
- For any $n \in \mathcal{N}$, define $1_n = [o \ldots o]$, the type taking n parameters.
- For $i = 1 \ldots n$, define $\pi_i^n = (\lambda u_1 \ldots u_n)\, u_i : 1_n$.
- Define $\Sigma^A : A \to 1_m$ to be $(\lambda f\, u_1 \ldots u_m)\, f(Ku_1) \ldots (Ku_m)$, for any type $A = [A_1 \ldots A_m]$.

We shall write K, π_i and Σ when these are not ambiguous. Note that there are exactly $n{+}2$ equivalence classes of type 1_n, namely the classes of $K\top$, $K\bot$ and the π_i.

Definition 2 A term $t : [A_1 \ldots A_m]$ is said to be ***i*-strict** if $\Sigma\, t$ is equivalent to π_i^m, and **constant** if $\Sigma\, t$ is equivalent to $K\top$ or to $K\bot$.

Note that any t has a normal form which is either $(\lambda \overline{x})\, \top$ or $(\lambda \overline{x})\, \bot$, or is in the form $(\lambda x_1 \ldots x_k)\, x_i \overline{s}$. In the first two cases, t is constant, while in the latter case, t is i-strict. $\square$

Definition 3 We define the **nested multiset order** (NMO) for types. This is essentially the usual nested multiset order, where we equate the type $[A_1 \ldots A_m]$ with the multiset $\{A_1, \ldots, A_m\}$. The NMO is defined to be the least binary relation $\ll$ such that if

$$A_1, \ldots, A_m \;\ll\; B_j,$$

then

$$[B_1 \ldots B_{j-1}\, A_1 \ldots A_m\, B_{j+1} \ldots B_n] \;\ll\; [B_1 \ldots B_n].$$

It is well known [**?**] that the nested multiset order is well-founded, *i.e.*, there are no infinite sequences $A_0 \gg A_1 \gg \cdots$. $\square$

2 The Algorithm

We now informally present our algorithm, along with an informal proof of correctness.

The listing algorithm proceeds by calculating certain combinators which are used to build up the complete lists. The combinators will be in the following form: given a type $A = [A_1 \dots A_m]$, we shall compute, for $i = 1 \dots m$, finite lists of types $\langle A\rangle_{i,1} \dots \langle A\rangle_{i,k_i^A}$ and combinators $C_i^A : \langle A\rangle_{i,1} \to \dots \to \langle A\rangle_{i,k_i^A} \to A$ with the following properties:

(1) $\langle A\rangle_{i,j} \ll A$ in the nested multi-set order.

(2) Each i-strict equivalence class of type A has a representative in the form

$$C_i^A(t_1)\dots(t_{k_i^A}).$$

Given these combinators, a listing algorithm can be given quite simply:

Lemma 4 *If there is an algorithm computing combinators satisfying (1) and (2) above, then there is a listing algorithm.*

Proof: The following is a recursive procedure for computing a complete list of equivalence class representatives, using combinators as given above.

Given a type A, (recursively) compute complete lists at each type $\langle A\rangle_{i,j}$. Then a complete list at type A is given by $K\top$, $K\bot$ and the terms in the forms listed in (2) above, where the t_j are taken from the appropriate complete lists just computed.

The recursion is finite due to the well-foundedness of the NMO. $\square$

It thus remains to find an algorithm that given a type A, computes the k_i^A, the types $\langle A\rangle_{i,j}$ and the combinators C_i^A. We shall also find combinators $D_{i,j}^A$ that are inverse to the C_i^A in the sense that if $r : A$ is i-strict, then

$$r \equiv C_i^A(D_{i,1}^A r)\dots(D_{i,k_i^A}^A r). \tag{3}$$

We shall abbreviate the RHS above to $C_i^A(\overline{D}_i^A r)$. Note that (3) implies (2).

3 Calculation of Combinators

We shall now derive the required data satisfying (1–3). We use induction on types. The induction step is that we are given the C_i^A etc., and we wish to find C_j^B, where $B = [B_1 \dots B_n]$ with $B_j = A = [A_1 \dots A_m]$.

Let $s : B$ be j-strict. We show that for any $\overline{r} : \overline{B}$, we have

$$\begin{aligned} s\,\overline{r} \quad \equiv \quad & \Sigma\, r_j \,\big(F_1\, r_1 \ldots r_{j-1}\, (\overline{D}^A_1 r_j)\, r_{j+1} \ldots r_n\big) \\ & \qquad \ldots \,\big(F_m\, r_1 \ldots r_{j-1}\, (\overline{D}^A_m r_j)\, r_{j+1} \ldots r_n\big) \end{aligned} \tag{4}$$

where

$$F_i\, r_1 \ldots r_{j-1}\, d_1 \ldots d_{k^A_i}\, r_{j+1} \ldots r_n \;\equiv\; s\, r_1 \ldots r_{j-1}\, (C^A_i\, \overline{d})\, r_{j+1} \ldots r_n. \tag{5}$$

To prove (4), note that for i-strict r_j, the $\Sigma\, r_j$ in (4) selects its ith parameter, which by (5) is equivalent to

$$s\, r_1 \ldots r_{j-1}\, \big(C^A_i(\overline{D}^A_i\, r_j)\big)\, r_{j+1} \ldots r_n.$$

As r_j is i-strict, this is equivalent to $s\,\overline{r}$ by (3). For constant r_j, (4) is easily verified using the j-strictness of s.

We therefore define C^B_j to be

$$\begin{aligned} & \big(\lambda\overline{\phi}\big)\, \big(\lambda\overline{y}\big)\, \Sigma\, y_j\, \big(\phi_1\, y_1 \ldots y_{j-1}\, (\overline{D}^A_1 y_j)\, y_{j+1} \ldots y_n\big) \\ & \qquad\qquad \ldots\, \big(\phi_m\, y_1 \ldots y_{j-1}\, (\overline{D}^A_m y_j)\, y_{j+1} \ldots y_n\big) \end{aligned}$$

and define $D^B_{j,i}$ to be

$$\big(\lambda\psi\big)\, \big(\lambda y_1 \ldots y_{j-1}\, u_1 \ldots u_{k^A_i}\, y_{j+1} \ldots y_n\big)\, \psi\, y_1 \ldots y_{j-1}\, (C^A_i\, \overline{u})\, y_{j+1} \ldots y_n.$$

By (4) and (5), this gives $C^B_j(\overline{D}^B_j\, s) \equiv s$ for j-strict $s : B$, so that (2) and (3) hold for B. The types $\langle B\rangle_{j,i}$ are

$$[B_1 \ldots B_{j-1}\, \langle A\rangle_{i,1} \ldots \langle A\rangle_{i,k^A_i}\, B_{j+1} \ldots B_n].$$

As $\langle A\rangle_{i,x} \ll A$, we have also $B_{j,i} \ll B$, so that (1) holds also.