Defining data structures via Böhm-out[1]

(To Corrado Böhm on the occasion of his 70th birthday)

ENRICO TRONCI

Dipartimento Matematica Pura ed Applicata, Università di L'Aquila,[2]
Via Vetoio, Coppito, 67100 L'Aquila, Italy
(*e-mail*: tronci@smaq20.univaq.it)

Abstract

We show that any recursively enumerable subset of a data structure can be regarded as the solution set to a Böhm-out problem.

Capsule review

If **D** is the data structure determined by a free algebra, then an element d in **D** can be represented in a natural way by a lambda term $\bar{d}$. The paper gives a representation of the data structure itself by constructing a single lambda term $\langle\langle \mathbf{D} \rangle\rangle$ such that M represents an element of the data structure **D** iff M is a solution of the following two equations:

$$\langle\langle \mathbf{D} \rangle\rangle \; \mathbf{I} M = M$$
$$\langle\langle \mathbf{D} \rangle\rangle \; (Kz) M = z.$$

This follows from corollary 4.4.

Statman (1989) had proved already that every recursively enumerable set of closed lambda terms which is closed under β-conversion is the solution set of a combinatory equation. However, for the given data structures the solution above is without coding.

1 Introduction

It is well known that data structures can be specified using anarchic heterogeneous term algebras (e.g. see Böhm and Berarducci, 1985, and Leivant, 1983. For example, the natural numbers are the universe of the term algebra **Nat** = $\langle$\{Nat\}, \{s : Nat → Nat, 0 : Nat\}$\rangle$ (i.e. Nat = \{0, s(0), s(s(0)), ...\}). Moreover, such specification yields immediately a representation in the λ-calculus for the constructors and the elements of the data structure.

Is it possible to represent the set of elements of a data structure as the solution set to a system of equations in the λ-calculus?

Statman (1989) showed that any β-closed and recursively enumerable (RE) set of closed λ-terms is the solution set to a combinator equation. Thus the answer to the

[1] This paper is a revised version of Tronci (1991a, Section 3).
[2] This paper was written when the author was a Post-Doct at LIP ENS Lyon, 46 Allée d'Italie, 69364 Lyon cedex 07, France.

https://doi.org/10.1017/S0956796800001234 Published online by Cambridge University Press

above question is affirmative. However, the equation in Statman (1989) involves coding and is difficult to study.

We show that if we restrict ourselves to data structures the situation can be considerably improved in the following sense.

Let $\mathbf{D}$ be a data structure, $d \in \mathbf{D}$, $\bar{d} \in \Lambda^0$ a closed λ-term representing d and $\bar{\mathbf{D}} = \{\bar{d} \mid d \in \mathbf{D}\}$. Then (4.3) there is a closed λ-term $\langle\langle \mathbf{D} \rangle\rangle$ s.t. ($I \equiv (\lambda t.\, t)$):

$$[\forall d \in \mathbf{D} \langle\langle \mathbf{D} \rangle\rangle z \bar{d} = z\bar{d}] \quad \text{and}$$
$$\forall M \in \Lambda^0 [\langle\langle \mathbf{D} \rangle\rangle (\lambda a.\, z) M = z \Rightarrow \exists d \in \mathbf{D} [\langle\langle \mathbf{D} \rangle\rangle \mathbf{I} M = \bar{d}]].$$

Thus the solution set to the Böhm-out problem specified by the equation $\langle\langle \mathbf{D} \rangle\rangle (\lambda a.\, z) X = z$ (unknown X) can be regarded as $\bar{\mathbf{D}}$ (up to the *filtering* $\langle\langle \mathbf{D} \rangle\rangle \mathbf{I}$) and the λ-terms representing the elements of the data structure $\mathbf{D}$ are exactly the λ-terms Böhming-out z from $\langle\langle \mathbf{D} \rangle\rangle (\lambda a.\, z)$. Moreover, the definition of the λ-term $\langle\langle \mathbf{D} \rangle\rangle$ closely follows the definition (type structure) of $\mathbf{D}$. Hence, in some sense, $\langle\langle \mathbf{D} \rangle\rangle$ is the definition inside the λ-calculus of the type definition of the data structure $\mathbf{D}$.

As an easy corollary we have (4.8) that any RE subset of a data structure can be regarded (up to the filtering $\langle\langle \mathbf{D} \rangle\rangle \mathbf{I}$) as the solution set to a Böhm-out problem.

These results do not achieve the full strength of Statman (1989) since, in general, for each $d \in \mathbf{D}$ there are infinitely many (non-β-convertible) $M \in \Lambda^0$ s.t.: $\langle\langle \mathbf{D} \rangle\rangle (\lambda a.\, z)\, M = z$ and $\langle\langle \mathbf{D} \rangle\rangle \mathbf{I} M = \bar{d}$. This, however, does not yield any problem since the data 'coded' in M can be read computing $\langle\langle \mathbf{D} \rangle\rangle \mathbf{I} M$. On the contrary, this feature suggests a uniform approach to modularity (since no reference is made to the structure of M) and to data compression (since M can be much smaller than $\bar{d} = \langle\langle \mathbf{D} \rangle\rangle \mathbf{I} M$).

2 Basic definitions

We assume the reader to be familiar with Barendregt (1984) of which, unless otherwise stated, we use notations and conventions. Var is the set of variables of Λ.

Let $\mathscr{L} \subseteq \Lambda$. The elements of Form$(\mathscr{L}) = \{M = N \mid M, N \in \mathscr{L}\}$ are called formulas (on $\mathscr{L}$). From the context, it will be easy to decide if $M = N$ is a formula or stands for $M =_\beta N$.

Here are a few *famous* λ-terms: $\mathbf{I} \equiv \lambda t.\, t$: $\mathbf{U}_i^n \equiv \lambda t_1 \ldots t_n.\, t_i$, where $1 \leqslant i \leqslant n$.

If f: $A \to B$ is a partially defined function and $a \in A$, we write: $f(a) = \downarrow$ (or $f(a)\downarrow$) for 'f is defined' and $f(a) = \uparrow$ (or $f(a)\uparrow$) for 'f is undefined'.

When defining a set by induction we omit the part '... the smallest set ...'.

If L is a list we write $\alpha \in L$ for α occurs in the list L and $L =$ nil for L is empty.

In the presence of binding operators, we assume that there are no name clashings (*hygiene condition* as that in Barendregt, 1984, Section 2.1.13).

Definition 2.1 0. *A system of (equations) is a pair* (Γ, X) *where* Γ *is a finite set of formulas on* Λ *(the equations) and* X *is a finite subset of* Var *(the unknowns).*
1. *Let* $\mathscr{S} = (\Gamma, X)$ *be a system. A formula* $M = N \in \Gamma$ *is said to be an equation of* $\mathscr{S}$. *By abuse of language we write also* $M = N \in \mathscr{S}$. *A variable* $x \in X$ *is said to be an unknown of* $\mathscr{S}$. *Unless otherwise stated, equations are considered up to* β*-conversion,*

https://doi.org/10.1017/S0956796800001234 Published online by Cambridge University Press

e.g. $\mathscr{S}_1 = (\{\lambda z.xz)\, z = x\}, \{x\}) = (\{x\, z = x\}, \{x\})$ *is a system with one equation and one unknown (namely* x*).*

2. *A system* $\mathscr{S} = (\Gamma, \{x_1, \ldots, x_n\})$ *is said to be* β*-solvable iff there exist* $D_1, \ldots D_n \in \Lambda^0$ s.t. *for all equations* $M = N$ *in* $\mathscr{S}$ *we have* $M[x_1 := D_1, \ldots, x_n := D_n] =_\beta N[x_1 := D_1, \ldots x_n := D_n]$ *(i.e.* M *and* N *are made* β*-convertible by replacing the variables* $x_1, \ldots x_n$ *with closed* λ*-terms* $D_1, \ldots D_n$*).*

The substitution $D[\,] \equiv (\lambda x_1, \ldots x_n.[\,])\, D_1 \ldots D_n$ (s.t. $D[Q] = Q[x_1 := D_1, \ldots x_n := D_n]$) *is said to be a* β*-solution for* $\mathscr{S}$*.*

Solutions $(\mathscr{S}) = \{(D_1, \ldots D_n) \in (\Lambda^0)^n \mid D[\,] \equiv (\lambda x_1 \ldots x_n.[\,])\, D_1 \ldots D_n$ *is a* β*-solution for* $\mathscr{S}\}$.

3. *An SL-system (separation-like system) is a system* $\mathscr{S} = (\Gamma, X)$ *with equations having form (up to* β*-conversion)* $x\vec{M} = z$ *where* $x \in X$ *and* $z \notin X$*, e.g.* $\mathscr{S}_2 = (\{x\, z = z\}, \{x\})$ *is an SL-system (with unique solution* **I**)*, but* $\mathscr{S}_1$ *is not an SL-system. Thus an SL-system is a (simultaneous) Böhm,-out problem. SL-systems are studied in Tronci (1991).* □

We give the definition of *data structure*. All the definitions in (2.2) come (or are inspired) from Böhm and Berarducci (1985), Leivant (1983) and Burris and Sankappanavar (1981).

Definition 2.2 0. GndType $= \{a_0, a_1, \ldots\}$ *is the set of* ground types *(or basic types). We use the letters* A, B*, with or without subscripts, as syntactic variables of* GndType.

1. *The set* Type *(of* types*) is defined as follows*: GndType ⊆ Type; *if* $A_1, \ldots A_n$, $B \in$ GndType *then* $A_1 \times \ldots \times A_n \to B \in$ Type.

We use the letters α, σ, γ*, with or without subscripts, as syntactic variables on* Type.

2. *A* function declaration *is a pair* f:α*, where* f *is any alphanumeric identifier and* $\alpha \in$ Type.

3. *A* language *(or* type*)* of data F *is a finite list of pairwise distinct function declarations, i.e. if* f_1:α_1, f_2:α_2 *are in* F *then* f_1 *and* f_2 *are distinct identifiers and* α_1, α_2 *are distinct types.*

4. *Let* F *be a language of data and* $A \in$ GndType*.* A *is said to be a parameter in* F *iff* [A *occurs in* F *and there is no function declaration in* F *having form* f: $A_1 \times \ldots \times A_n \to A$]. A *is said to be a nonparameter in* F *iff* [A *occurs in* F *and* A *is not a parameter in* F].

5. *Let* F *be a language of data with exactly* k ($k \geqslant 0$) *parameters and* Card$(F) = n$.

0. *We denote with* $F(1), \ldots F(k)$ *the parametric types of* F *and with* $F(k+1), \ldots F(n)$ *the nonparametric types of* F*. Since the order in which types are considered matters we choose it as follows:*

$F(i) =$ **case** $1 \leqslant i \leqslant k$ **then** *the* i*-th parameter encountered scanning* F;
$k < i \leqslant n$ **then** *the* $(i-k)$*-th nonparameter encountered scanning* F;
end.

1. *We define*: *univ*$(F) = \langle F(1), \ldots F(n)\rangle$, param$(F) = \langle F(1), \ldots F(k)\rangle$, nonpar$(F) = \langle F(k+1), \ldots F(n)\rangle$ *(see the example below, point* 2).

6. *The set DS of data structures is defined as follows: if* F *is a language of data s.t.* param(F) *is empty then* $F \in DS$*; if* F *is a language of data s.t.* Card(param(F)) $= k > 0$ *and* $\mathbf{D}_1, \ldots \mathbf{D}_k \in DS$ *then* $((\Lambda F(1) \ldots F(k).F)\, \mathbf{D}_1 \ldots \mathbf{D}_k) \in DS$.

(*Here* Λ *has nothing to do with* λ*-conversion or polymorphic* λ*-calculus.*)

https://doi.org/10.1017/S0956796800001234 Published online by Cambridge University Press

A data structure of language F is a data structure $\mathbf{D}$ *having form* $(k = \mathrm{Card}(\mathrm{param}(F)))\ ((\Lambda F(1)\dots F(k).F)\,\mathbf{D}_1\dots\mathbf{D}_k)$ *(the case* $k = 0$ *is allowed and then* $\mathbf{D} = F$*).*

7. An assignment of generators $\mathcal{g}$ *is a map that assigns to each element* $A \in \mathrm{GndType}$ *a set* $\mathcal{g}(A)$ *s.t.* $\forall A, B \in \mathrm{GndType}\,[A \neq B \Rightarrow \mathcal{g}(A) \cap \mathcal{g}(B) = \varnothing]$.

We write $\mathcal{g} \in GA$ *for* $\mathcal{g}$ *is an assignment of generators.*

8. *Let F be a language of data,* $\mathcal{g} \in GA$ *and* $A \in \mathrm{univ}(F)$.

0. *The set* $T(F, \mathcal{g}, A)$ *of terms of type A* in *F is defined as follows:* $\mathcal{g}(A) \cup \{f \mid f: A \in F\} \subseteq T(F, \mathcal{g}, A)$; *if* $f: A_1 \times \dots \times A_n \to A \in F$ *and* $\forall i \in \{1, \dots n\}[p_i \in T(F, \mathcal{g}, A_i)]$ *then* $f(p_1, \dots p_n) \in T(F, \mathcal{g}, A)$.

1. $T(F, \mathcal{g}) = \cup\{T(F, \mathcal{g}, A) \mid A \in \mathrm{nonpar}(F)\}$.

9. *Let* $\mathbf{D} = ((\Lambda : F(1)\dots F(k).F)\,\mathbf{D}_1\dots\mathbf{D}_k) \in DS$ *and* $\mathrm{Card}(\mathrm{univ}(F)) = n$. *The function* $[\![\mathbf{D}]\!]$ *with domain* $\{0, 1, \dots n\}$ *is defined as follows (we write* $[\![\mathbf{D}, i]\!]$ for $[\![\mathbf{D}]\!](i)$*):*

$$[\![\mathbf{D}, i]\!] = \textbf{case } i = 0 \quad \textbf{then } ([\![\mathbf{D}, k+1]\!] \cup \dots \cup [\![\mathbf{D}, n]\!]);$$
$$1 \leqslant i \leqslant n \textbf{ then } T(F, \mathcal{g}, F(i)); \textbf{ end},$$

where: $\mathcal{g}(A) = \textbf{if } A = F(j) \in \mathrm{param}(F) \textbf{ then } [\![\mathbf{D}_j, 0]\!] \textbf{ else } \varnothing$.

$[\![\mathbf{D}, 0]\!]$ *defines the set of elements of a data structure* $\mathbf{D}$. *This set is obtained instancing (via* $T(F, \mathcal{g}, F(i))$*) the parameters of F (i.e.* $F(1), \dots F(k)$*) with the elements of the parametric data structures of* $\mathbf{D}$ *(i.e.* $\mathbf{D}_1, \dots \mathbf{D}_k$*). We write* $|\mathbf{D}|$ *for* $[\![\mathbf{D}, 0]\!]$ and $d \in \mathbf{D}$ *for* $f \in |\mathbf{D}|$ *and we call d an element of* $\mathbf{D}$. □

Example 0. The following are types: $a_1, a_1 \times a_0 \times a_{13} \to a_7$.

1. The following are function declarations: cons: $A \times L \to L$, nil: L.

2. $F = \langle \mathrm{egxp}: L \times A \times B \to L, \mathrm{gp}: L \rangle$ is a language of data s.t. $\mathrm{univ}(F) = \langle A, B, L \rangle$, $\mathrm{param}(F) = \langle A, B \rangle$, $\mathrm{nonpar}(F) = \langle L \rangle$.

3. $\mathbf{D} = ((\Lambda A.\langle \mathrm{cons}: A \times L \to L, \mathrm{nil}: L \rangle)\langle s: N \to N, 0: N \rangle)$ is a data structure (representing lists over natural numbers).

4. Let $Boole = \langle \mathrm{tt}: B, \mathrm{ff}: B \rangle$ be a language of data and $\mathbf{Boole} = Boole$ a data structure. Then: $|\mathbf{Boole}| = [\![\mathbf{Boole}, 0]\!] = [\![\mathbf{Boole}, 1]\!] = \{tt, ff\}$.

5. Let $Nat = \langle s: N \to 0: N \rangle$ be a language of data and $\mathbf{Nat} = Nat$ a data structure. Then: $|\mathbf{Nat}| = [\![\mathrm{Nat}, 0]\!] = [\![\mathbf{Nat}, 1]\!] = \{0, s(0), s(s(0)), \dots\}$.

6. Let $L = \langle \mathrm{cons}: A \times L \to L, \ \mathrm{nil}: L \rangle$ be a language of data and $\mathbf{List(Nat)} = ((\Lambda : A.L)\,\mathbf{Nat})$ be a data structure (i.e. the one in point 3 above). Then: $|\mathbf{List(Nat)}| = [\![\mathbf{List}, 0]\!] = [\![\mathbf{List}, 2]\!] = \{\mathrm{nil}, \mathrm{cons}(0, \mathrm{nil}), \dots\}$ and $[\![\mathbf{List}, 1]\!] = |\mathbf{Nat}|$. □

Notation Let $\mathbf{D}_1, \dots \mathbf{D}_{n+1} \in DS$. A function f from $\mathbf{D}_1 \times \dots \times \mathbf{D}_n$ to $\mathbf{D}_{n+1}$ (written $f: \mathbf{D}_1 \times \dots \times \mathbf{D}_n \to \mathbf{D}_{n+1}$) is a function from $|\mathbf{D}_1| \times \dots \times |\mathbf{D}_{n+1}|$ to $|\mathbf{D}_{n+1}|$. Moreover, we write $d \in \mathbf{D}_1 \times \dots \times \mathbf{D}_n\,((d_1, \dots d_n) \in \mathbf{D}_1 \times \dots \times \mathbf{D}_n)$ for $d \in |\mathbf{D}_1| \times \dots \times |\mathbf{D}_n|\,((d_1, \dots d_n) \in |\mathbf{D}_1| \times \dots \times |\mathbf{D}_n|)$. □

3 Representing data structures in the λ-calculus

Given a data structure $\mathbf{D}$ (see Definition 2.2(6)) we want to represent its elements in the λ-calculus. This can be done in many ways (e.g. see Böhm and Berarducci, 1985; Tronci, 1991, Section 8.2), but the representation given here has nice properties for what follows.

https://doi.org/10.1017/S0956796800001234 Published online by Cambridge University Press

Any element of a data structure can be represented in the λ-calculus.

Definition 3.1 *Let* **D** *be a data structure of language* $F = \langle b_i : F(h(i, 1)) \times \ldots \times F(h(i, q(i))) \to F(h(i, q(i)+1)) \mid i = (1, \ldots m) \rangle$ *with* Card (univ (F)) $= N$. *We define*:

0. $\overline{b_i} \equiv \lambda t_1 \ldots t_{q(i)}\, b_1 \ldots b_m . b_i t_1 \ldots t_{q(i)}, i \in \{1, \ldots m\}$.
1. $\overline{b_{i,j}} \equiv \lambda t . t \Omega_1 \ldots \Omega_{i-1} \mathbf{U}_j^{q(i)} \Omega_{i+1} \ldots \Omega_m, i \in \{1, \ldots m\}, j \in \{1, q(i)\}$.
2. $\overline{\text{case}^F} \equiv \overline{\text{case}^{\mathbf{D}}} \equiv \lambda t . t \mathbf{U}^{q(1)+m}_{q(1)+1} \mathbf{U}^{q(2)+m}_{q(2)+2} \ldots \mathbf{u}^{q(m)+m}_{q(m)+m}$.
   When there is no ambiguity we will also write $\overline{\text{case}}$ for $\overline{\text{case}^F}$.
3. *Let* $b(t_1, \ldots t_r) \in \mathbf{D}$. *We define*: $\overline{b(t_1, \ldots t_r)} = \overline{bt_1} \ldots \overline{t_r}$.
4. $\bar{\mathbf{D}} = \{\bar{d} \mid d \in \mathbf{D}\}$. □

Example 0. Let **Boole** be as earlier. Then $\overline{tt} = \lambda b_1\, b_2 . b_1$, $\overline{ff} = \lambda b_1\, b_2 . b_2$.
1. Let **Nat** be as earlier. Then $\bar{s} = \lambda t b_1 b_2 . b_1 t, \bar{0} = \lambda b_1 b_2 . b_2$, $\overline{s_{1,1}} = \bar{p} = \lambda t . t \mathbf{U}_1^2$, $\overline{\text{case}} = \lambda t . t \mathbf{U}_2^3 \mathbf{U}_2^2$, $\overline{s(0)} = \overline{s\bar{0}} = \lambda b_1 b_2 . \bar{0}$, etc.
2. Let **List(Nat)** be as earlier. Then: $\overline{\text{cons}} = \lambda t_1 t_2 b_1 b_2 . b_1 t_1 t_2$, $\overline{\text{nil}} = \lambda b_1 b_2 . b_2$, $\overline{\text{cons}(0, \text{nil})} = \lambda b_1 b_2 . b_1 \bar{0}\, \overline{\text{nil}}$, etc.

Proposition 3.2 *Let* F *and* **D** *be as in Definition* 3.1. $\forall i \in \{1, \ldots m\} \forall j \in \{1, \ldots q(i)\}$ *we have*:

0. $\overline{b_{i,j}}(\overline{b_i} t_1 \ldots t_{q(i)}) = t_j$.
1. $\overline{\text{case}}(\overline{b_i} t_1 \ldots t_{q(i)}) = \mathbf{U}_i^m$.

Proof 0. $\overline{b_{i,j}}(\overline{b_i} t_1 \ldots t_{q(i)}) = \overline{b_i} t_1 \ldots t_{q(i)} \Omega_1 \ldots \Omega_{i-1} \mathbf{U}_j^{q(i)} \Omega_{i+1} \ldots \Omega_m = \mathbf{U}_j^{q(i)} t_1 \ldots t_{q(i)} = t_j$.
1. $\overline{\text{case}}(\overline{b_i} t_1 \ldots t_{q(i)}) = \overline{b_i}\, t_1 \ldots t_{q(i)} \mathbf{U}^{q(1)+m}_{q(1)+1} \mathbf{U}^{q(2)+m}_{q(2)+2} \ldots \mathbf{U}^{q(m)+m}_{q(m)+m} = \mathbf{U}^{q(i)+m}_{q(i)+i}\, t_1 \ldots t_{q(i)} = \mathbf{U}_i^m$. □

4 Data structures as solutions to SL-systems

The elements of a data structure **D** can be represented inside the λ-calculus as in Definition 3.1; however, such a technique does not yield a definition of **D** *inside* the λ-calculus. Can we find a λ-term that somehow represents $\bar{\mathbf{D}}$? In Statman (1989) we find a positive answer to such a question. More specifically, it is proved that given a β-closed and recursively enumerable set of closed λ-terms $\mathscr{G}$ it is possible to find a closed λ-term G s.t.: $M \in \mathscr{G}$ iff $GM = G$. However, the construction of G involves coding; thus the equation $Gx = G$ is very difficult to study. We show (Section 4.8) that if we restrict ourselves to data structures an easy and natural (without coding) representation can be found. In particular, we show that a data structure **D** can be represented as the set of solutions to a Böhm-out problem, i.e. to an SL-system. However, unlike usual representations, each element of **D** will have infinitely many λ-terms representing it.

Example 4.1 0. Let **Boole** be as earlier. Define: $\langle\langle \mathbf{Boole} \rangle\rangle = \lambda zt . t(z \overline{tt})(z \overline{ff})$, $\mathscr{S}_B = (\{\langle\langle \mathbf{Boole} \rangle\rangle \quad (\lambda a . z) x = z\}, \quad \{x\}) = (\{xzz = z\}, \quad \{x\})$. Thus $\overline{\mathbf{Boole}} = \{\overline{tt}, \overline{ff}\} = \{\mathbf{U}_1^2, \mathbf{U}_2^2\} \subseteq$ Solutions $(\mathscr{S}_B)$ and $\forall M \in$ Solutions $(\mathscr{S}_B)[M = \overline{tt}$ or $M = \overline{ff}]$. Note that $[\forall d \in \mathbf{Boole}\ \langle\langle \mathbf{Boole} \rangle\rangle z\bar{d} = z\bar{d}]$ and $\forall M \in \Lambda^0$ $[\langle\langle \mathbf{Boole} \rangle\rangle (\lambda a . z) M = z \Rightarrow \exists d \in \mathbf{Boole}[\langle\langle \mathbf{Boole} \rangle\rangle \mathbf{I} M = \bar{d}]]$. Thus, up to β-conversion, Solutions $(\mathscr{S}_B)$ can be regarded as $\overline{\mathbf{Boole}}$.

https://doi.org/10.1017/S0956796800001234 Published online by Cambridge University Press

1. Let **Nat** be as in earlier and $\langle\langle \mathbf{Nat}\rangle\rangle$ any closed λ-term satisfying the equation $\langle\langle \mathbf{Nat}\rangle\rangle = \lambda zt.t(\langle\langle \mathbf{Nat}\rangle\rangle(\lambda t.z(\bar{s}t)))(z\bar{0})$ (e.g. obtained using the construction in Barendregt, 1984, Section 6.5.2).

The Böhm tree for $\langle\langle \mathbf{Nat}\rangle\rangle$ is shown in Fig. 1.

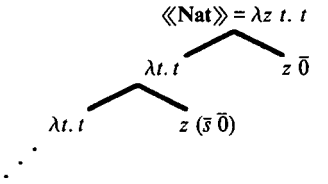

Fig. 1.

Define: $\mathscr{S}_N = (\{\langle\langle \mathbf{Nat}\rangle\rangle(\lambda a.z)x = z\}, \{x\})$. It is easy to check that $\overline{\mathbf{Nat}} \subseteq \text{Solutions}(\mathscr{S}_N)$. Moreover, we have: $[\forall n \in \mathbf{Nat}\langle\langle \mathbf{Nat}\rangle\rangle z\bar{n} = z\bar{n}]$ and $\forall M \in \Lambda^0[\langle\langle \mathbf{Nat}\rangle\rangle(\lambda a.z)M = z \Rightarrow \exists n \in \mathbf{Nat}[\langle\langle \mathbf{Nat}\rangle\rangle \mathbf{I}M = \bar{n}]]$. Thus if M is a solution to $\mathscr{S}_N$ then we can regard M as a representation of the (unique) natural number n s.t. $\langle\langle \mathbf{Nat}\rangle\rangle \mathbf{I}M = \bar{n}$. Moreover, for each natural number n there is a solution to $\mathscr{S}_N$ representing n (namely $\bar{n}$). Note, however, that unlike **Boole**, each element of **Nat** has infinitely many non-β-convertible representations, e.g. let $Z \equiv \lambda ab.a\mathbf{U}_3^3 b$. Then $Z \in \text{Solutions}(\mathscr{S}_N)$, $\langle\langle \mathbf{Nat}\rangle\rangle \mathbf{I}z = \bar{0}$ (i.e. Z represents 0) and $\forall N \in \overline{\mathbf{Nat}}[Z \neq N]$. The λ-term $\langle\langle \mathbf{Nat}\rangle\rangle \mathbf{I}$ maps any solution to $\mathscr{S}_N$ into an element of $\overline{\mathbf{Nat}}$. This allows us to regard any solution to $\mathscr{S}_N$ as (a representation for) an element of **Nat**.

2. Let **List(Nat)** be as earlier and $\langle\langle \mathbf{List(Nat)}\rangle\rangle$ any closed λ-term satisfying the equation $\langle\langle \mathbf{List(Nat)}\rangle\rangle = \lambda zt.t(\langle\langle \mathbf{Nat}\rangle\rangle(\lambda t_1.\langle\langle \mathbf{List(Nat)}\rangle\rangle(\lambda t_2.z(\overline{\mathbf{cons}}\, t_1\, t_2))))(z\,\overline{\mathbf{nil}})$.

The Böhm tree for $\langle\langle \mathbf{List(Nat)}\rangle\rangle$ is as shown in Fig. 2.

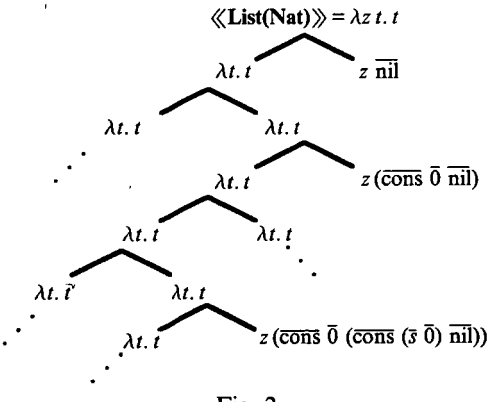

Fig. 2.

Define: $\mathscr{S}_L = (\{\langle\langle \mathbf{List(Nat)}\rangle\rangle(\lambda a.z)x = z\}, \{x\})$. We have: $\overline{\mathbf{List(Nat)}} \subseteq \text{Solutions}(\mathscr{S}_L)$. Moreover, we have: $[\forall d \in \mathbf{List(Nat)}\langle\langle \mathbf{List(Nat)}\rangle\rangle z\bar{d} = z\bar{d}]$ and $\forall M \in \Lambda^0[\langle\langle \mathbf{List(Nat)}\rangle\rangle(\lambda a.z)M = z \Rightarrow \exists d \in \mathbf{List(Nat)}[\langle\langle \mathbf{List(Nat)}\rangle\rangle \mathbf{I}M = \bar{d}]]$. Thus if M is a solution to $\mathscr{S}_L$ then we can regard M as a representation of

https://doi.org/10.1017/S0956796800001234 Published online by Cambridge University Press

the (unique) list d s.t. $\langle\langle\mathbf{List(Nat)}\rangle\rangle\,\mathbf{I}\,M = \bar{d}$. Moreover, for each list d there is a solution to $\mathscr{S}_L$ representing d (namely $\bar{d}$). Note, however, that each element of **List** has infinitely many non-β-convertible representations, e.g. let Z be as in 4.1(1). Then $Z \in \text{Solutions}(\mathscr{S}_N), \langle\langle\mathbf{List(Nat)}\rangle\rangle\,\mathbf{I}\,Z = \overline{\text{nil}}$ (i.e. Z represents nil) and $\forall L \in \overline{\mathbf{List(Nat)}}[Z \neq L]$. The λ-term $\langle\langle\mathbf{List(Nat)}\rangle\rangle\,\mathbf{I}$ maps any solution to $\mathscr{S}_L$ into an element of $\overline{\mathbf{List(Nat)}}$. This allows us to regard any solution to $\mathscr{S}_L$ as (a representation for) an element of **List(Nat)**.

Example 4.1 leads to the following general definition.

Definition 4.2 *Let* **D** *and* F *be as in Section* 3 *with* $\text{Card}(\text{param}(F)) = k\,(\geqslant 0)$. 0. *We denote with* $\langle\langle\mathbf{D}\rangle\rangle$ *any function from* $\{0, 1, \ldots n\}$ *to* Λ^0 *satisfying the following equations* (*we write* $\langle\langle\mathbf{D}, i\rangle\rangle$ *for* $\langle\langle\mathbf{D}\rangle\rangle(i)$):

$$\forall i \in \{0, 1, \ldots n\}\,\langle\langle\mathbf{D}, i\rangle\rangle = \lambda z t.\, t(\langle\langle b_1, i\rangle\rangle z)\ldots(\langle\langle b_m, i\rangle\rangle z);$$

where: $\forall j \in \{1, \ldots m\}\,\forall i \in \{1, \ldots n\}$

$$\begin{aligned}\langle\langle b_j, 0\rangle\rangle = \lambda z.\,\langle\langle \mathbf{G}_{h(j,1)}, a(h(j,1))\rangle\rangle(\lambda t_1.\,\langle\langle \mathbf{G}_{h(j,2)},\\ \times a(h(j,2))\rangle\rangle(\lambda t_2 \ldots(\lambda t_{q(j)-1}.\,\langle\langle \mathbf{G}_{h(j,q(j))},\\ \times a(h(j,q(j)))\rangle\rangle(\lambda t_{q(j)}.\, z(b_j\, t_1 \ldots t_{q(j)})))\ldots);\end{aligned}$$

$$\langle\langle b_j, i\rangle\rangle = \textbf{if } h(j, q(j)+1) = i \textbf{ then } \langle\langle b_j, 0\rangle\rangle$$

else $\lambda z.\,\Omega$; (*the role of* Ω *is illustrated in the example below*)

$$a(i) = \textbf{if } 1 \leqslant i \leqslant k \textbf{ then } 0 \textbf{ else } i;\ \mathbf{G}_i = \textbf{if } 1 \leqslant i \leqslant k \textbf{ then } \mathbf{D}_i \textbf{ else } \mathbf{D}.$$

The λ-*terms* $\langle\langle\mathbf{D}, i\rangle\rangle$ *can be found, e.g. using the construction in Barendregt* (1984, *Section* 6.5.2).

In the following we write $\langle\langle\mathbf{D}\rangle\rangle$ for $\langle\langle\mathbf{D}, 0\rangle\rangle$. This ambiguity will be harmless.
1. $\mathscr{S}_D = (\{\langle\langle\mathbf{D}\rangle\rangle(\lambda a.\,z)\,x = z\}, \{x\}]$. □

Example 4.3 0. The λ-terms $\langle\langle\mathbf{Boole}\rangle\rangle$, $\langle\langle\mathbf{Nat}\rangle\rangle$ and $\langle\langle\mathbf{List(Nat)}\rangle\rangle$ in Section 4 are constructed according to the definition previously given.
1. Let $\mathit{Woods} = \langle \text{bic}: A \times F \to T,\ \text{join}: F \times T \to F,\ \text{emp}: F\rangle$ and $\mathbf{Woods} = ((\Lambda A.\,\mathit{Woods})\,\mathbf{Nat})$ (**Nat** as earlier). We have:

$$\begin{aligned}\langle\langle\mathbf{Woods}, 0\rangle\rangle &= \lambda z t.\, t(\langle\langle\mathbf{Nat}, 0\rangle\rangle(\lambda t_1.\,\langle\langle\mathbf{Woods}, 2\rangle\rangle(\lambda t_2.\, z\,(\overline{\text{bic}}\, t_1\, t_2))))\\ &\quad\times(\langle\langle\mathbf{Woods}, 2\rangle\rangle(\lambda t_1.\,\langle\langle\mathbf{Woods}, 3\rangle\rangle(\lambda t_2.\, z\,(\overline{\text{join}}\, t_1\, t_2))))\\ &\quad\times(z\,\overline{\text{emp}});\\ \langle\langle\mathbf{Woods}, 1\rangle\rangle &= \lambda z t.\, t\Omega\Omega\Omega;\\ \langle\langle\mathbf{Woods}, 2\rangle\rangle &= \lambda z t.\, t\Omega(\langle\langle\mathbf{Woods}, 2\rangle\rangle(\lambda t_1.\,\langle\langle\mathbf{Woods}, 3\rangle\rangle(\lambda t_2.\, z\,(\overline{\text{join}}\, t_1\, t_2))\\ &\quad\times(z\,\overline{\text{emp}});\\ \langle\langle\mathbf{Woods}, 3\rangle\rangle &= \lambda z t.\, t(\langle\langle\mathbf{Nat}, 0\rangle\rangle(\lambda t_1.\,\langle\langle\mathbf{Woods}, 2\rangle\rangle(\lambda t_2.\, z\,(\overline{\text{bic}}\, t_1\, t_2))))\,\Omega\Omega.\end{aligned}$$

Woods is the data structure defining trees (type T and set of terms $[\![\mathbf{Woods}, 2]\!]$) and forests (type F and set of terms $[\![\mathbf{Woods}, 3]\!]$) over the parameter **Nat**.

Roughly speaking, we *represent* sets via Böhm-out properties. Thus, intuitively,

https://doi.org/10.1017/S0956796800001234 Published online by Cambridge University Press

any λ-term Böhming-out z from $(\langle\langle \mathbf{Woods}, i\rangle\rangle z)$ represents an element of the set $[\![\mathbf{Woods}, i]\!]$ $(i = 0, \ldots 3)$. More specifically, the following properties are satisfied: for each $M \in \Lambda^0[\langle\langle \mathbf{Woods}, i\rangle\rangle (\lambda a.z)\, M = z \Rightarrow \exists\, d \in [\![\mathbf{Woods}, i]\!][\langle\langle \mathbf{Woods}, i\rangle\rangle z\, M = z\bar{d}]]$; for each $d \in [\![\mathbf{Woods}, i]\!]\, \exists\, M \in \Lambda^0[\langle\langle \mathbf{Woods}, i\rangle\rangle z\, M = z\bar{d}]$.

The sets $[\![\mathbf{Woods}, i]\!]$ are always defined as the union of the codomains of the appropriate constructors, e.g. $[\![\mathbf{Woods}, 0]\!]$ (being the union of $[\![\mathbf{Woods}, 2]\!]$ and $[\![\mathbf{Woods}, 3]\!]$) is the union of the codomains of bic, join and emp. The codomain of bic is formed with terms having form $\mathrm{bic}(t_1, t_2)$, where t_1 is in **Nat** (i.e. $[\![\mathbf{Nat}, 0]\!]$) and t_2 is in $[\![\mathbf{Woods}, 2]\!]$ (i.e. t_2 has type F). Such a set is represented with the λ-term $\langle\langle \mathrm{bic}, 0\rangle\rangle = \lambda z.\langle\langle \mathbf{Nat}, 0\rangle\rangle (\lambda t_1.\langle\langle \mathbf{Woods}, 2\rangle\rangle (\lambda t_2.z\,(\overline{\mathrm{bic}}\, t_1\, t_2)))$. An analogous reasoning leads to the definition of $\langle\langle \mathrm{join}, 0\rangle\rangle$ and $\langle\langle \mathrm{emp}, 0\rangle\rangle$. A λ-term represents an element of $[\![\mathbf{Woods}, 0]\!]$ iff it represents an element in the codomain of bic or join or emp. Thus a λ-term M represents an element of $[\![\mathbf{Woods}, 0]\!]$ iff M can Böhm-out z from $(\langle\langle \mathrm{bic}, 0\rangle\rangle z)$ or from $(\langle\langle \mathrm{join}, 0\rangle\rangle z)$ or from $(\langle\langle \mathrm{emp}, 0\rangle\rangle z)$. Thus we define: $\langle\langle \mathbf{Woods}, 0\rangle\rangle = \lambda zt.t(\langle\langle \mathrm{bic}, 0\rangle\rangle z)(\langle\langle \mathit{join}, 0\rangle\rangle z)(\langle\langle \mathrm{emp}, 0\rangle\rangle z)$. The set $[\![\mathbf{Woods}, 1]\!]$ is a parameter and no constructor has its codomain contained in $[\![\mathbf{Woods}, 1]\!]$. Thus $\langle\langle \mathbf{Woods}, 1\rangle\rangle = \lambda zt.t\Omega\Omega\Omega$ (thus Ω represents the empty set). The set $[\![\mathbf{Woods}, 2]\!]$ is the union of the codomains of join and emp. Reasoning as before yields the definition for $\langle\langle \mathbf{Woods}, 2\rangle\rangle$. The set $[\![\mathbf{Woods}, 3]\!]$ is the codomain of bic. Reasoning as before yields the definition for $\langle\langle \mathbf{Woods}, 3\rangle\rangle$.

Note that the sets $[\![\mathbf{Woods}, 2]\!]$ (trees) and $[\![\mathbf{Woods}, 3]\!]$ (forests) are not themselves data structures. Thus, for example, lists of trees (forests) do not form a data structure but are a subset of the data structure list of woods. This is in agreement with the fact that to define a function f by recursion on trees (forests) we have to define f by recursion on woods (i.e. trees and forests).

2. Languages of data not defining anything are also handled. Let $\mathit{Empty} = \langle \mathrm{empty}: E \to E\rangle$ and $\mathbf{Empty} = \mathit{Empty}$. Then $\langle\langle \mathbf{Empty}, 0\rangle\rangle = \lambda zt.t(\langle\langle \mathbf{Empty}, 0\rangle\rangle (\lambda a.z\,(\overline{\mathrm{empty}}\, a)))$ =

$\lambda zt.t$

|

$\lambda t.t$

|

$\lambda t.t$

|

Thus $\mathrm{Solutions}((\{\langle\langle \mathbf{Empty}\rangle\rangle (\lambda a.z)\, x = z\}, \{x\})) = \varnothing = \overline{\mathbf{Empty}}$.

Up to the filtering $\langle\langle \mathbf{D}\rangle\rangle \mathbf{I}$, $\mathrm{Solutions}(\mathscr{S}_D)$ can be regarded as $\bar{\mathbf{D}}$.

Proposition 4.3 *Let* **D** *be a data structure.*

0. $\forall\, d \in \mathbf{D}[\langle\langle \mathbf{D}\rangle\rangle z\bar{d} = \bar{d}]$.

1. $\forall\, M \in \Lambda^0[\langle\langle \mathbf{D}\rangle\rangle (\lambda a.z)\, M = z \Rightarrow \exists\, d \in \mathbf{D}[\langle\langle \mathbf{D}\rangle\rangle \mathbf{I}\, M = \bar{d}]]$.

Proof Intuitive justifications are given in the previous example. Here are more formal proofs.

Let **D** and F be as in Section 3 with $\mathrm{Card}(\mathrm{param}(F)) = k\ (\geqslant 0)$. Let $d \in \mathbf{D}$.

0. The proof proceeds by induction on $\mathrm{size}(d)$ (= the number of function symbols in d).

https://doi.org/10.1017/S0956796800001234 Published online by Cambridge University Press

Case 0. size$(d) = 1$. Then $d = b_i$ where $b_i : F(h(i,1))$ is in F. Thus $\overline{d} = \overline{b_i} \equiv \lambda b_1 \dots b_m . b_i$. We have:

$$\langle\langle \mathbf{D} \rangle\rangle z\overline{d} = \langle\langle \mathbf{D}, 0 \rangle\rangle z\overline{d} = (\lambda z t . t(\langle\langle b_1, 0 \rangle\rangle z) \dots (\langle\langle b_m, 0 \rangle\rangle z)) z\overline{d}$$
$$= \overline{d}(\langle\langle b_1, 0 \rangle\rangle z) \dots (\langle\langle b_m, 0 \rangle\rangle z) = \langle\langle b_i, 0 \rangle\rangle z = (\lambda z . z\overline{b_i}) z = z\overline{b_i}.$$

Case 1. size$(d) > 1$. Let $d = b_i(d_1, \dots d_{q(i)})$ where $b_i : F(h(i,1)) \times \dots \times F(h(i, q(i))) \to F(h(i, q(i)+1))$ is in F. Thus $\overline{d} = \overline{b_i}\,\overline{d_1} \dots \overline{d_{q(i)}} = \lambda b_1 \dots b_m . b_i\,\overline{d_1} \dots \overline{d_{q(i)}}$. We have:

$$\langle\langle \mathbf{D} \rangle\rangle z\overline{d} = d(\langle\langle\langle\langle b_1, 0 \rangle\rangle z) \dots (\langle\langle b_m, 0 \rangle\rangle z) = \langle\langle b_i, 0 \rangle\rangle z\,\overline{d_1} \dots \overline{d_{q(i)}}$$
$$= \langle\langle \mathbf{G}_{h(i,1)}, a(h(i,1)) \rangle\rangle (\lambda t_1 . \langle\langle \mathbf{G}_{h(i,2)}, a(h(i,2)) \rangle\rangle (\lambda t_2 \dots (\lambda t_{q(i)-1} .$$
$$\times \langle\langle \mathbf{G}_{h(i,q(i))}, a(h(i,q(i))) \rangle\rangle (\lambda t_{q(i)} . z(b_i t_1 \dots t_{q(i)})) \dots) \overline{d_1} \dots \overline{d_{q(i)}}.$$

Moreover, size$(d_1) <$ size(d).

Case 1.0. $1 \leqslant h(i,1) \leqslant k$. Thus $F(h(i,1))$ is a parameter and d_1 is in $\mathbf{D}_{h(i,1)}$, hence $\langle\langle \mathbf{G}_{h(i,1)}, a(h(i,1)) \rangle\rangle = \langle\langle \mathbf{D}_{h(i,1)}, 0 \rangle\rangle$. We have:

$$\langle\langle \mathbf{D} \rangle\rangle z\overline{d} = \langle\langle \mathbf{D}_{h(i,1)}, 0 \rangle\rangle (\lambda t_1 . \langle\langle \mathbf{G}_{h(i,2)}, a(h(i,2)) \rangle\rangle (\lambda t_2 \dots (\lambda t_{q(i)-1} .$$
$$\times \langle\langle \mathbf{G}_{h(i,q(i))}, a(h(i,q(i))) \rangle\rangle (\lambda t_{q(i)} . z(\overline{b_i} t_1 \dots t_{q(i)})) \dots) \overline{d_1} \dots \overline{d_{q(i)}}$$
$$= \langle\langle \mathbf{D}_{h(i,1)} \rangle\rangle (\lambda t_1 \dots) \overline{d_1} \dots \overline{d_{q(i)}}$$
$$= (\lambda t_1 \dots) \overline{d_1} \dots \overline{d_{q(i)}} \text{ (by induction hypothesis).}$$

Thus $\langle\langle \mathbf{D} \rangle\rangle z\overline{d} = \langle\langle \mathbf{G}_{h(i,2)}, a(h(i,2)) \rangle\rangle (\lambda t_2 \dots (\lambda t_{q(i)-1} . \langle\langle \mathbf{G}_{h(i,q(i))}, a(h(i,q(i))) \rangle\rangle$ $(\lambda t_{q(i)} . z(\overline{b_i}\,\overline{d_1}\, t_2 \dots t_{q(i)})) \dots) d_2 \dots \overline{d_{q(i)}}$. Using the induction hypothesis repeatedly (to take care of $\overline{d_2} \dots \overline{d_{q(i)}}$ we have: $\langle\langle \mathbf{D} \rangle\rangle z\overline{d} = z(\overline{b_i}\,\overline{d_1}\,\overline{d_2} \dots \overline{d_{q(i)}}) = z\overline{d}$.

Case 1.1. $h(i,1) > k$. Thus $F(h(i,1))$ is not a parameter, d_1 is in $\mathbf{D}$, and there is a j s.t. $h(i,1) = h(j, q(j)+1)$ and $d_1 = b_j(d_{1,1} \dots d_{1,q(j)})$. Hence $\overline{d_1} = \overline{b_j}\,\overline{d_{1,1}} \dots \overline{d_{1,q(j)}} = \lambda b_1 \dots b_m . b_j\,\overline{d_{1,1}} \dots \overline{d_{1,q(j)}}$. Moreover $\langle\langle \mathbf{G}_{h(i,1)}, a(h(i,1)) \rangle\rangle = \langle\langle \mathbf{D}, h(i,1) \rangle\rangle = \langle\langle \mathbf{D}, h(j, q(j)+1) \rangle\rangle$. We have:

$$\langle\langle \mathbf{D} \rangle\rangle z\,\overline{d} = \langle\langle \mathbf{D}, h(j,q(j)+1) \rangle\rangle (\lambda t_1 . \langle\langle \mathbf{G}_{h(i,2)}, a(h(i,2)) \rangle\rangle (\lambda t_2 \dots (\lambda t_{q(i)-1} .$$
$$\times \langle\langle \mathbf{G}_{h(i,q(i))}, a(h(i,q(i))) \rangle\rangle (\lambda t_{q(i)} . z(\overline{b_i} t_1 \dots t_{q(i)})) \dots) \overline{d_1} \dots \overline{d_{q(i)}}$$
$$= (\lambda z t . t\langle\langle b_1, h(j,q(j)+1) \rangle\rangle z) \dots (\langle\langle b_m, h(j,q(j)+1) \rangle\rangle z)) (\lambda t_1 \dots) \overline{d_1} \dots \overline{d_{q(i)}}$$
$$= \overline{d_1}(\langle\langle b_1, h(j,q(j)+1) \rangle\rangle (\lambda t_1 \dots)) \dots (\langle\langle b_m, h(j,q(j)+1) \rangle\rangle (\lambda t_1 \dots)) \overline{d_2} \dots \overline{d_{q(i)}}.$$

By Definition 4.2(0) we have: $\langle\langle b_j, h(j,q(j)+1) \rangle\rangle = \langle\langle b_j, 0 \rangle\rangle$. Thus:

$$\overline{b_j}\,\overline{d_{1,1}} \dots \overline{d_{1,q(j)}} (\langle\langle b_1, h(j,q(j)+1) \rangle\rangle (\lambda t_1 \dots)) \dots (\langle\langle b_m, h(j,q(j)+1) \rangle\rangle$$
$$\times (\lambda t_1 \dots)) \overline{d_2} \dots \overline{d_{q(i)}})$$
$$= (\langle\langle b_j, 0 \rangle\rangle (\lambda t_1 \dots)) \overline{d_{1,1}} \dots \overline{d_{1,q(j)}}\, \overline{d_2} \dots \overline{d_{q(i)}}$$
$$= \langle\langle \mathbf{D}, 0 \rangle\rangle (\lambda t_1 \dots) \overline{d_1} \dots \overline{d_{q(i)}}$$
$$= (\lambda t_1 \dots) \overline{d_1} \dots \overline{d_{q(i)}}$$

(by induction hypothesis).

Thus $\langle\langle \mathbf{D} \rangle\rangle z\overline{d} = \langle\langle \mathbf{G}_{h(i,2)}, a(h(i,2)) \rangle\rangle (\lambda t_2 \dots (\lambda t_{q(i)-1} . \langle\langle \mathbf{G}_{h(i,q(i))}, a(h(i,q(i))) \rangle\rangle$ $(\lambda t_{q(i)} . z(\overline{b_i}\,\overline{d_1}\, t_2 \dots t_{q(i)})) \dots) \overline{d_2} \dots \overline{d_{q(i)}}$. Hence using the induction hypothesis repeatedly (to take care of $\overline{d_2} \dots \overline{d_{q(i)}}$) we have: $\langle\langle \mathbf{D} \rangle\rangle z\overline{d} = z(\overline{b_i}\,\overline{d_1}\,\overline{d_2} \dots \overline{d_{q(i)}}) = z\overline{d}$.

https://doi.org/10.1017/S0956796800001234 Published online by Cambridge University Press

1. Notation: if $M \in \Lambda$ we define head(M) = **if** $M = \lambda \vec{y}.z\vec{L}$ **then** z **else** $\uparrow$.

We prove that $\forall \alpha \in \mathrm{Seq}$ [if head$((\langle\langle \mathbf{D}\rangle\rangle z)_\alpha) = z$ then $\exists d \in \mathbf{D}$ s.t. $(\langle\langle \mathbf{D}\rangle\rangle z)_\alpha = z\bar{d}$]. Examples are given in Example 4.1. From this property the thesis follows. In fact, let $M \in \Lambda^0$ s.t. $\langle\langle \mathbf{D}\rangle\rangle (\lambda a.z) M = z$. Then there exists $H \in \Lambda$ s.t. $\langle\langle \mathbf{D}\rangle\rangle zM = zH$. Thus there exists $\alpha \in \mathrm{Seq}$ s.t.: head$((\langle\langle \mathbf{D}\rangle\rangle z)_\alpha) = z$ and M Böhm-outs $(\langle\langle \mathbf{D}\rangle\rangle z)_\alpha$ from $\langle\langle \mathbf{D}\rangle\rangle z$, i.e. $\langle\langle \mathbf{D}\rangle\rangle zM \to_\beta (\langle\langle \mathbf{D}\rangle\rangle z)_\alpha \dots \to_\beta zH$. Hence there exists $d \in \mathbf{D}$ s.t. $(\langle\langle \mathbf{D}\rangle\rangle z)_\alpha = z\bar{d}$ and $\langle\langle \mathbf{D}\rangle\rangle zM \to_\beta (z\bar{d}) \dots \to_\beta zH$. This implies $H = \bar{d}$ and the thesis follows.

The proof goes by induction on length(α). Let α s.t. head$((\langle\langle \mathbf{D}\rangle\rangle z)_\alpha) = z$. Then $\alpha = \langle j\rangle^*\theta$ and $0 \leqslant j < m$. Let $i = j+1$. We have $\langle\langle \mathbf{D}\rangle\rangle z = \lambda t.t(\langle\langle b_1, 0\rangle\rangle z) \dots (\langle\langle b_m, 0\rangle\rangle z)$. Thus

$$(\langle\langle \mathbf{D}\rangle\rangle z)_\alpha = (\langle\langle b_i, 0\rangle\rangle z)_\theta = (\langle\langle \mathbf{G}_{h(i,1)}, a(h(i,1))\rangle\rangle (\lambda t_1.\langle\langle \mathbf{G}_{h(i,2)}, a(h(i, \times 2))\rangle\rangle (\lambda t_2 \dots (\lambda t_{q(i)-1}.\langle\langle \mathbf{G}_{h(i,q(i))}, a(h(i,q(i)))\rangle\rangle (\lambda t_{q(i)}.z(\bar{b}_i t_1 \dots t_{q(i)}))) \dots))_\theta.$$

Case 2. $\theta = \langle\,\rangle$. Then $q(i) = 0$ and $(\langle\langle \mathbf{D}\rangle\rangle z)_\alpha = (\langle\langle b_i, 0\rangle\rangle z) = z\bar{b}_i$.

Case 1. $\theta \neq \langle\,\rangle$ and $1 \leqslant h(i,1) \leqslant k$. Thus $\langle\langle \mathbf{G}_{h(i,1)}, a(h(i,1))\rangle\rangle = \langle\langle \mathbf{D}_{h(i,1)}, 0\rangle\rangle$. We have:

head$((\langle\langle \mathbf{D}\rangle\rangle z)_\alpha) = z$ and $(\langle\langle \mathbf{D}\rangle\rangle z)_\alpha = (\langle\langle \mathbf{D}_{h(i,1)}, 0\rangle\rangle (\lambda t_1.\langle\langle \mathbf{G}_{h(i,2)}, a(h(i, 2))\rangle\rangle (\lambda t_2 \dots (\lambda t_{q(i)-1}.\langle\langle \mathbf{G}_{h(i,q(i))}, a(h(i,q(i)))\rangle\rangle (\lambda t_{q(i)}.z(\bar{b}_i t_1 \dots t_{q(i)}))) \dots))_\theta.$

Thus there exists $\sigma < \theta$ s.t.: $\theta = \sigma^*\rho$ and head$((\langle\langle \mathbf{D}_{h(i,1)}, 0\rangle\rangle z)_\sigma) = z$. This, by induction hypothesis, implies that there exists $d_1 \in \mathbf{D}$ s.t. $(\langle\langle \mathbf{D}_{h(i,1)}, 0\rangle\rangle z)_\sigma = z\bar{d}_1$. Thus:

$$\begin{aligned}
&(\langle\langle \mathbf{D}_{h(i,1)}, 0\rangle\rangle (\lambda t_1.\langle\langle \mathbf{G}_{h(i,2)}, a(h(i,2))\rangle\rangle (\lambda t_2 \dots (\lambda t_{q(i)-1}.\langle\langle \mathbf{G}_{h(i,q(i))}, a(h(i, \\
&\times q(i)))\rangle\rangle (\lambda t_{q(i)}.z(\bar{b}_i t_1 \dots t_{q(i)}))) \dots))_{(\sigma^*\rho)} \\
&= ((\lambda t_1.\langle\langle \mathbf{G}_{h(i,2)}, a(h(i,2))\rangle\rangle (\lambda t_2 \dots (\lambda t_{q(i)-1}.\langle\langle \mathbf{G}_{h(i,q(i))}, a(h(i,q(i)))\rangle\rangle (\lambda t_{q(i)}.z(\bar{b}_i t_1 \\
&\times \dots t_{q(i)}))) \dots)\bar{d}_1)_\rho \\
&= (\langle\langle \mathbf{G}_{h(i,2)}, a(h(i,2))\rangle (\lambda t_2 \dots (\lambda t_{q(i)-1}.\langle\langle \mathbf{G}_{h(i,q(i))}, a(h(i,q(i)))\rangle\rangle (\lambda t_{q(i)}.z(\bar{b}_i \bar{d}_1 t_2 \dots \\
&\times t_{q(i)}))) \dots))_\rho.
\end{aligned}$$

Hence, using the induction hypothesis repeatedly (to get rid of $\mathbf{G}_{h(i,2)}, \dots \mathbf{G}_{h(i,q(i))}$) we have the thesis.

Case 2. $\theta \neq \langle\,\rangle$ and $h(i,1) > k$. Thus $\langle\langle \mathbf{G}_{h(i,1)}, a(h(i,1))\rangle\rangle = \langle\langle \mathbf{D}, i\rangle\rangle$. We have:

$$\begin{aligned}
&(\langle\langle \mathbf{D}\rangle\rangle z)_\alpha = (\langle\langle b_i, 0\rangle\rangle z)_\theta = (\langle\langle \mathbf{D}, i\rangle\rangle (\lambda t_1.\langle\langle \mathbf{G}_{h(i,2)}, a(h(i,2))\rangle\rangle (\lambda t_2 \dots (\lambda t_{q(i)-1}. \\
&\times \langle\langle \mathbf{G}_{h(i,q(i))}, a(h(i,q(i)))\rangle\rangle (\lambda t_{q(i)}.z(\bar{b}_i t_1 \dots t_{q(i)}))) \dots))_\theta \\
&= ((\lambda z t.t(\langle\langle b_1, i\rangle\rangle z) \dots (\langle\langle b_m, i\rangle\rangle z))(\lambda t_1 \dots))_\theta \\
&= (\lambda t.t(\langle\langle b_1, i\rangle\rangle (\lambda t_1 \dots)) \dots (\langle\langle b_m, i\rangle\rangle (\lambda t_1 \dots)))_\theta.
\end{aligned}$$

By Definition 4.2 for $r = 1, \dots m$ we have: $\langle\langle b_r, i\rangle\rangle =$ **if** $h(r, q(r)+1) = i$ **then** $\langle\langle b_r, 0\rangle\rangle$ **else** $\lambda z.\Omega$. Since head$((\langle\langle \mathbf{D}\rangle\rangle z)_\alpha) = z$ we have: $\theta = \langle r-1\rangle^*\sigma$, $h(r, q(r)+1) = i$ and $0 < r \leqslant m$. Thus:

$$(\langle\langle \mathbf{D}\rangle\rangle z)_\alpha = (\langle\langle b_r, 1\rangle\rangle (\lambda t_1 \dots))_\sigma = (\langle\langle b_r, 0\rangle\rangle (\lambda t_1 \dots))_\sigma = (\langle\langle \mathbf{D}, 0\rangle\rangle (\lambda t_1 \dots))_\theta.$$

Since head$((\langle\langle \mathbf{D}\rangle\rangle z)_\alpha) =$ head$((\langle\langle \mathbf{D}, 0\rangle\rangle (\lambda t_1 \dots))_\theta) = z$ there exists $\tau < \theta$ s.t.: $\theta = \tau^*\rho$ and head$((\langle\langle \mathbf{D}, 0\rangle\rangle z)_\tau) = z$. Thus, by induction hypothesis, there exists

https://doi.org/10.1017/S0956796800001234 Published online by Cambridge University Press

$d_1 \in \mathbf{D}$ s.t. $(\langle\langle \mathbf{D}, 0\rangle\rangle z)_\tau = z\overline{d_1}$. Thus $(\langle\langle \mathbf{D}\rangle\rangle z)_\alpha = (\langle\langle \mathbf{D}, 0\rangle\rangle(\lambda t_1 \ldots))_\theta = ((\lambda t_1 \ldots)\overline{d_1})_\rho = (\langle\langle \mathbf{G}_{h(i,2)}, a(h(i,2))\rangle\rangle(\lambda t_2 \ldots (\lambda t_{q(i)-1} . \langle\langle \mathbf{G}_{h(i,q(i))}, a(h(i,q(i)))\rangle\rangle (\lambda t_{q(i)} . z(\overline{b_i}\,\overline{d_1}\, t_2 \ldots t_{q(i)})))\ldots))_\rho$. Hence, using the induction hypothesis repeatedly (to get rid of $\mathbf{G}_{h(i,2)}, \ldots \mathbf{G}_{h(i,q(i))}$) the thesis follows. □

Corollary 4.4 *Let* $\mathbf{D}$ *be a data structure. Then*:
0. $\forall d \in \mathbf{D}[\langle\langle \mathbf{D}\rangle\rangle(\lambda a . z)\,\overline{d} = z]$.
1. $\forall d \in \mathbf{D}[\langle\langle \mathbf{D}\rangle\rangle \mathbf{I}\overline{d} = \overline{d}]$.
2. $\bar{\mathbf{D}} \subseteq \text{Solutions}(\mathscr{S}_D)$.
3. $\forall M \in \text{Solutions}(\mathscr{S}_D)\, \exists d \in \mathbf{D}[\langle\langle \mathbf{D}\rangle\rangle \mathbf{I}M = \overline{d}]$.

Proof 0. Let $d \in \mathbf{D}$. We have (by 4.3.0): $\langle\langle \mathbf{D}\rangle\rangle(\lambda a . z)\,\overline{d} = (\lambda a . z)\,\overline{d} = z$.
1. *Let* $d \in \mathbf{D}$. We have (by 4.3.0): $\langle\langle \mathbf{D}\rangle\rangle \mathbf{I}\overline{d} = \mathbf{I}\overline{d} = \overline{d}$.
2. From 4.4.0.
3. It is just a restatement of Proposition 4.3(1). □

Propositions 4.3 and 4.4 suggest the following definition:

Definition 4.5 *Let* $\mathbf{D}$ *be a data structure and* $d \in \mathbf{D}$.
0. $M \in \Lambda^0$ *is said to represent an element of* $\mathbf{D}$ (*or, by abuse of language, to be an element of* $\mathbf{D}$) *iff* $[\langle\langle \mathbf{D}\rangle\rangle(\lambda a . z)\,M = z]$. *We write* $M \in \mathbf{D}$ *for* M *is an element of* $\mathbf{D}$.
1. $M \in \Lambda^0$ *is said to represent* $d \in \mathbf{D}$ (*notation*: $rep(\mathbf{D}, M) = d$) *iff* $[M \in \mathbf{D}$ *and* $\langle\langle \mathbf{D}\rangle\rangle \mathbf{I}M = \overline{d}]$. *By Proposition* 4.3(1) $rep(\mathbf{D}, M)$ *is well defined.*

Equation $\langle\langle \mathbf{D}\rangle\rangle(\lambda a . z)\,M = z$ defines the set of λ-terms representing elements of $\mathbf{D}$. This is the set $\{M \in \Lambda^0 \mid \langle\langle \mathbf{D}\rangle\rangle(\lambda a . z)\,M = z\}$. However, given a λ-term M representing an element of $\mathbf{D}$ how can we know which element of $\mathbf{D}M$ is representing? This question is answered by the λ-term $(\langle\langle \mathbf{D}\rangle\rangle \mathbf{I})$. In fact, by Proposition 4.3(1), for each $M \in \mathbf{D}$ there exists (a unique) $d \in \mathbf{D}$ s.t. $\langle\langle \mathbf{D}\rangle\rangle \mathbf{I}M = \overline{d}$. Thus we can use $\langle\langle \mathbf{D}\rangle\rangle \mathbf{I}$ to *read* which element of $\mathbf{D}$ the λ-term $M \in \mathbf{D}$ is representing.

This state of affairs is quite different from usual notions of representation for data structures. In fact, for each $d \in \mathbf{D}$ we have infinitely many non β-convertible $M \in \Lambda^0$ representing d, i.e. s.t. $\langle\langle \mathbf{D}\rangle\rangle \mathbf{I}M = \overline{d}$, (e.g. see Example 4.1(1, 2) whereas usual notions of representation for data structures assign exactly one λ-term (up to β-conversion) to each $d \in \mathbf{D}$. However, this 'many-representatives' situation is harmless since for each $M \in \Lambda^0$ representing $d \in \mathbf{D}$ we have $\langle\langle \mathbf{D}\rangle\rangle \mathbf{I}M = \overline{d}$. Thus using $\langle\langle \mathbf{D}\rangle\rangle \mathbf{I}$ we can map all the λ-terms representing d to $\overline{d}$.

Alternatively, we could prune the set of possible representations from the beginning. To do this it is sufficient to replace $[\langle\langle \mathbf{D}\rangle\rangle(\lambda a . z)\,M = z]$ in 4.5.0 with $[\langle\langle \mathbf{D}\rangle\rangle(\lambda a . z)\,M = z$ and $\langle\langle \mathbf{D}\rangle\rangle \mathbf{I}M = M]$.

From a mathematical point of view, these two approaches are equivalent and it is just a matter of taste which one to follow. However, they are not equivalent from a computer science point of view. Suppose we want to represent a list of length 1000 having all elements equal to 0. With the first approach (many representatives) we could represent our list with a program that generates it (thus avoiding repeating the same information (i.e. 0) 1000 times). However, this possibility is ruled out by the second approach, since we would be forced to write down a list with 1000 elements

https://doi.org/10.1017/S0956796800001234 Published online by Cambridge University Press

(even though they are all equal). For this reason we prefer to adopt the first approach (i.e. Definition 4.5 as it was given). However, the reader that prefers to think of an element of a data structure as having exactly one representation (thus adopting the modified version of Definition 4.5(0)) can do so in reading the rest of the paper. The only change is that it will be no longer true that data structures are defined via Böhm-out, since the new version of Definition 4.5(0) involves a fixed point equation, namely $\langle\langle \mathbf{D} \rangle\rangle \mathbf{I} M = M$.

Propositions 4.3 and 4.4 show that any data structure $\mathbf{D}$ can be defined via the Böhm-out problem (SL-system) $\mathscr{S}_D$. In Proposition 4.6 we show that with such representation any partial recursive function can be represented.

Proposition 4.6 *Let* $\mathbf{D}_1, \dots \mathbf{D}_{n+1} \in \mathrm{DS}$ *and* $f: \mathbf{D}_1 \times \dots \times \mathbf{D}_n \to \mathbf{D}_{n+1}$ *be a partial recursive function. Then* $\exists F \in \Lambda^0 \forall M_1, \dots M_n \in \Lambda^0$ [*if* $\forall i \in \{1, \dots n\} M_i \in \mathbf{D}_i$ *then* $FM_1 \dots M_n$

$= \mathbf{if}\ f(\mathrm{rep}(\mathbf{D}_1, M_1), \dots \mathrm{rep}(\mathbf{D}_n, M_n)) \downarrow$
$\mathbf{then}\ \overline{f(\mathrm{rep}(\mathbf{D}_1, M_1), \dots \mathrm{rep}(\mathbf{D}_n, M_n))}$
$\mathbf{else}$ unsolvable].

Proof Using the data structure representation in Section 3 and defining f with a system of equations $\mathscr{S}$ (e.g. as sketched in Tronci, 1991, 0.0, 0.1, 8.2) we can obtain a closed λ-term $\bar{f}$ representing f by solving $\mathscr{S}$ (e.g. with the algorithm in Tronci 1991, 8.0). This is possible since we have a representation for the inverses of the constructors $(b_{i,j})$ and the case function. Defining $F = \lambda t_1 \dots t_n . \bar{f}(\langle\langle \mathbf{D}_1 \rangle\rangle \mathbf{I} t_1) \dots (\langle\langle \mathbf{D}_n \rangle\rangle \mathbf{I} t_n)$ the thesis follows by Proposition 4.3(1). □

Notation 4.7 Let f be as in Proposition 4.6. We write $\langle\langle f \rangle\rangle$ for F as constructed in the proof of 4.6. □

Any recursively enumerable (RE) subset of a data structure can be represented as the set of solutions to an SL-system.

Proposition 4.8 Let $\mathbf{D} \in \mathrm{DS}$ *and* $\mathscr{E}$ *be an* RE *subset of* $|\mathbf{D}|$. *Then there exists an* SL-*system* $\mathscr{S}$ s.t. $\forall M \in \Lambda^0$ [M *is a* β-*solution to* $\mathscr{S}$ *iff* [$M \in \mathbf{D}$ *and* $\mathrm{rep}(\mathbf{D}, M) \in \mathscr{E}$]].

Proof Let $f: \mathbf{D} \to \mathbf{Boole}$ a partial recursive function s.t. $\mathscr{E} = \mathrm{dom}(f)$ and $\forall d \in \mathbf{D}$ [$f(d) = tt$ or $f(d) = \uparrow$]. Define $\mathscr{S} = (\{\langle\langle \mathbf{D} \rangle\rangle (\lambda a . z) x = z, \langle\langle f \rangle\rangle xz\Omega = z\}, \{x\})$.

Let $M \in \Lambda^0$. We have: M is a β-solution to $\mathscr{S}$ iff [$\langle\langle \mathbf{D} \rangle\rangle (\lambda a . z) M = z$ and $\langle\langle f \rangle\rangle Mz\Omega = z$] iff [$M \in \mathbf{D}$ and $f(\mathrm{rep}(\mathbf{D}, M)) = tt$] iff [$M \in \mathbf{D}$ and $\mathrm{rep}(\mathbf{D}, M) \in \mathscr{E}$]. □

Example 4.9 0. Let **Boole** be as earlier and $\mathrm{True} = \{tt\} \subseteq |\mathbf{Boole}|$. The set True is defined by the SL-system $\mathscr{S}_{B,tt} = (\{\langle\langle \mathbf{Boole} \rangle\rangle (\lambda a . z) x = z, xz\Omega = z\}, \{x\}) = (\{xzz = z, xz\Omega = z), (x)\}$. Let $\mathrm{False} = \{ff\} \subseteq |\mathbf{Boole}|$. The set False is defined by the SL-system $\mathscr{S}_{B,ff} = (\{\langle\langle \mathbf{Boole} \rangle\rangle (\lambda a . z) x = z, x\Omega z = z\}, \{x\}) = (\{xzz = z, x\Omega z = z\}, \{x\})$.

1. Let **Nat** be as earlier and $f: \mathbf{Nat} \to \mathbf{Boole}$ a partial recursive function s.t. $\forall n \in \mathbf{Nat}$ [$f(n) = tt$ or $f(n) = \uparrow$]. The set $\mathrm{dom}(f)$ is defined by the SL-system $\mathscr{S}_{N,f} = (\{\langle\langle \mathbf{Nat} \rangle\rangle (\lambda a . z) x = z, \langle\langle f \rangle\rangle xz\Omega = z\}, \{x\})$. We have: $\forall M \in \Lambda^0$ [$M \in \mathrm{Solutions}(\mathscr{S}_{N,f})$ iff [$M \in \mathbf{Nat}$ and $f(\mathrm{rep}(\mathbf{Nat}, M)) = tt$]. □

Since equality over data structures is decidable systems of equations over data

https://doi.org/10.1017/S0956796800001234 Published online by Cambridge University Press

structures can be transformed into SL-systems (i.e. into a Böhm-out problem). An example will be sufficient to clarify the matter.

Example 4.10 Let $f,g: \mathbf{Nat} \to \mathbf{Nat}$ be total recursive functions. The equation $f(n) = g(n)$ (unknown n) can be transformed into an SL-system as follows. let Eq: $\mathbf{Nat} \times \mathbf{Nat} \to \mathbf{Boole}$ the equality function on **Nat**, i.e. the (recursive) function satisfying the equations: $\mathrm{Eq}(0,0) = tt$, $\mathrm{Eq}(0, s(n)) = ff$, $\mathrm{Eq}(s(m), 0) = ff$, $\mathrm{Eq}(s(m), s(n)) = \mathrm{Eq}(m,n)$. Let $\mathscr{S} = (\{\langle\langle \mathbf{Nat} \rangle\rangle (\lambda a.z)\, x = z, \langle\langle \mathrm{Eq} \rangle\rangle (\langle\langle f \rangle\rangle\, x)(\langle\langle g \rangle\rangle\, x)\, z\Omega = z\}, \{x\})$. Note that the first equation in $\mathscr{S}$ states that the unknown x is a natural number and the second represents the constraint $f(x) = g(x)$. We have: $\exists n \in \mathbf{Nat}\, [f(n) = g(n)]$ iff $\mathscr{S}$ has a β-solution. Moreover any solution to $\mathscr{S}$ yields a solution for the equation $f(n) = g(n)$, i.e.

$$\forall M \in \Lambda^0 [M \in \mathrm{Solutions}(\mathscr{S}) \Rightarrow [M \in \mathbf{Nat} \text{ and } f(\mathrm{rep}(\mathbf{Nat}, M)) = g(\mathrm{rep}(\mathbf{Nat}, M))]]. \quad \square$$

5 Conclusions

This paper shows that any RE subset of a data structure can be naturally specified as the solution set to an SL-system (i.e. to a Böhm-out problem). With this approach the type definition and the type checking algorithm for a data structure **D** come to be the same object (namely $\langle\langle \mathbf{D} \rangle\rangle$) and the elements of a data structure are exactly the β-solution to a Böhm-out problem (i.e. to the system $\mathscr{S}_D = (\{\langle\langle \mathbf{D} \rangle\rangle (\lambda a.z)\, x = z\}, \{x\})$). Defining a type α as the solution set to a system of equations yields a type definition independent from the structure of the λ-terms inhabiting α, since the only thing that matters is their operational behaviour (i.e. their ability to Böhm-out z). On the other hand, more traditional type systems rely on the structure of the λ-term to give a type judgement. Thus typing with equations seems to be a natural way to allow modular reasoning. Data compression is also easily accommodated, e.g. large numbers with a pattern on their digits could be represented with a program generating their digits, and this would still be accepted as a number.

Unfortunately, at this time the author does not know how to extend in a natural way this approach to functions over data structures.

Acknowledgements

I am grateful to Rick Statman for the helpful discussions we had on these topics and for his useful remarks and suggestions. I am also grateful to Peter B. Andrews and Corrado Böhm for their useful comments and suggestions about a preliminary version of this paper. Comments from an anonymous referee helped me to improve the presentation of this paper.

References

Barendregt, H. P. (1984) *The Lambda-calculus*. North-Holland.

Böhm, C. and Berarducci, A. (1985) Automatic synthesis of typed Λ-programs on term algebras. *Theoretical Computer Science*, **39**, 135–154.

Burris, S. and Sankappanavar, H. P. (1981) *A Course in Universal Algebra. Graduate Texts in Mathematics 78.* Springer-Verlag.

Leivant, D. (1981) Reasoning about functional programs and complexity classes associated with type disciplines. 24th Ann. Symp. on Foundation of Computer Science, pp. 460–469.

Statman, R. (1989) On sets of solutions to combinator equations. *Theoretical Computer Science*, **66**, 99–104.

Tronci, E. (1991a) Equational programming in λ-calculus. *Proc. LICS 91*, Amsterdam, July 15–18, IEEE Computer Society, pp. 191–202.

Tronci, E. (1991b) Equational programming in λ-calculus via SL-systems. PhD thesis, Department of Mathematics, Carnegie Mellon University.

https://doi.org/10.1017/S0956796800001234 Published online by Cambridge University Press