Higher Order Computability

John Longley & Dag Normann

March 28, 2022

Contents

1 Theory of Computability Models

1.1 Notations

- $e\downarrow$ 'the value of e is defined'
- $e\uparrow$ 'the value of e is undefined'
- $e = e'$ 'the values of both e and e' are defined and they are equal'

- $e \simeq e'$ 'if either e or e' is defined then so is the other and they are equal'
- $e \succeq e'$ 'if e' is defined then so is e and they are equal'

if e is a mathematical expression possibly involving the variable x, we write $\Lambda x.e$ to mean the ordinary (possibly partial) function f defined by $f(x) \simeq e$

Finite sequences of length n starts from index 0.

1.2 Higher-Order Computability Models

1.2.1 Computability Models

Definition 1.1. A **computability model C** over a set T of **type names** consists of

- an indexed family $|\mathbf{C}| = \{\mathbf{C}(\tau) \mid \tau \in \mathsf{T}\}$ of sets, called the **datatypes** of **C**
- for each $\sigma, \tau \in \mathsf{T}$, a set $\mathbf{C}[\sigma, \tau]$ of partial functions $f : \mathbf{C}(\sigma) \to \mathbf{C}(\tau)$, called the **operations** of **C**

s.t.

1. for each $\tau \in \mathsf{T}$, the identity function $\mathrm{id} : \mathbf{C}(\tau) \to \mathbf{C}(\tau)$ is in $\mathbf{C}(\tau, \tau)$
2. for any $f \in \mathbf{C}[\rho, \sigma]$ and $g \in \mathbf{C}[\sigma, \tau]$ we have $g \circ f \in \mathbf{C}[\rho, \tau]$ where $\circ$ denotes ordinary composition of partial functions

We shall use uppercase letters $A, B, C, ...$ to denote **occurrences** of sets within $|\mathbf{C}|$: that is, sets $\mathbf{C}(\tau)$ implicitly tagged with a type name τ. We shall write $\mathbf{C}[A, B]$ for $\mathbf{C}[\sigma, \tau]$ if $A = \mathbf{C}(\sigma)$ and $B = \mathbf{C}(\tau)$

In typical cases of interest, the operations of **C** will be 'computable' maps of some kind between datatypes

Definition 1.2. A computability model **C** is **total** if every operation $f \in \mathbf{C}[A, B]$ is a total function $f : A \to B$

Definition 1.3. A computability model **C** has **weak (binary cartesian) products** if there is an operation assigning to each $A, B \in |\mathbf{C}|$ a datatype $A \bowtie B \in |\mathbf{C}|$ along with operations $\pi_A \in \mathbf{C}[A \bowtie B, A]$ and $\pi_B \in \mathbf{C}[A \bowtie B, B]$ (known as **projections**) s.t. for any $f \in \mathbf{C}[C, A]$ and $g \in \mathbf{C}[C, B]$ there exists $\langle f, g \rangle \in \mathbf{C}[C, A \bowtie B]$ satisfying the following for all $c \in C$

1. $\langle f, g\rangle(c) \downarrow$ iff $f(c) \downarrow$ and $g(c) \downarrow$

2. $\pi_A(\langle f, g\rangle(c)) = f(c)$ and $\pi_B(\langle f, g\rangle(c)) = g(c)$

We say that $d \in A \bowtie B$ **represents** the pair (a, b) if $\pi_A(d) = a$ and $\pi_B(d) = b$

In contrast to the usual definition of categorical products, the operation $\langle f, g\rangle$ need not be unique, since many elements of $A \bowtie B$ may represent the same pair (a, b). We do not formally require that every (a, b) is represented in $A \bowtie B$, though in all cases of interest this will be so. The reader is also warned that $\pi_A \circ \langle f, g\rangle$ will not in general coincide with f .

Definition 1.4. A **weak terminal** in a computability model **C** consists of a datatype $I \in |\mathbf{C}|$ and an element $i \in I$ s.t. for any $A \in |\mathbf{C}|$ the constant function $\Lambda a.i$ is in $\mathbf{C}[A, I]$

If **C** has weak products and a weak terminal (I, i), then for any $A \in |\mathbf{C}|$ there is an operation $t_A \in \mathbf{C}[A, I \bowtie A]$ s.t. $\pi_A \circ t_A = \mathrm{id}_A$

1.2.2 Examples of Computability Models

Example 1.1. Model with single datatype $\mathbb{N}$ and whose operations $\mathbb{N} \rightharpoonup \mathbb{N}$ are precisely the Turing-computable partial functions. The model has standard products, since the well-known computable pairing operation

$$\langle m, n\rangle = (m + n)(m + n + 1)/2 + m$$

defines a bijection $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Any element $i \in \mathbb{N}$ may serve as a weak terminal, since $\Lambda n.i$ is computable

Example 1.2. untyped λ-calculus

Terms M of the λ-calculus are generated from a set of variable symbols x by means of the following grammar:

$$M ::= x \mid MM' \mid \lambda x.M$$

Writing L for the quotient set $\Lambda/ =_\beta$

We write $M[x \mapsto N]$ for the result of substituting N for all free occurrences of x within M

We define Λ to be the set of untyped λ-terms modulo α-equivalence.

Let $\sim$ be any equivalence relation on Λ with the following properties:

$$(\lambda x.M)N \sim M[x \mapsto N], \quad M \sim N \Rightarrow PM \sim PN$$

1. $(\lambda x.x)M \sim M$

2. If $M \sim N$, then $(\lambda x.N)M \sim (\lambda x.M)N$ and hence $N \sim M$.

3. If $M \sim N$ and $N \sim O$, then

Then we have $M \sim N \Rightarrow MP \sim NP$ since $(\lambda y.yP)M \sim (\lambda y.yP)N \Rightarrow MP \sim NP$.

As a example, we may define $=_\beta$ to be the smallest equivalence relation $\sim$ satisfying the above properties and also

$$M \sim N \Rightarrow \lambda x.M \sim \lambda x.N$$

Writing $[M]$ for the $\sim$-equivalence class of M, any term $P \in A$ induces a well-defined mapping $[M] \mapsto [PM]$ on $\Lambda/\sim$. The mappings induced by some P in this way are called **λ-definable**

We may regard $\Lambda/\sim$ as a total computability model: the sole datatype is $\Lambda/\sim$ itself, and the operations on it are exactly the λ-definable mappings. It also has weak products: a pair (M, N) may be represented by the term $pair\ M\ N$ where $pair = \lambda xyz.zxy$ the terms $fst = \lambda p.p(\lambda xy.x)$ and $snd = \lambda p.p(\lambda xy.y)$. We can check that $fst(pair\ M\ N) \sim M$ and $snd(pair\ M\ N) \sim N$

We can also obtain a submodel $\Lambda^0/\sim$ consisting of the equivalence classes of closed terms M

Example 1.3. Let B be any family of **base sets**, and let $\langle B\rangle$ denote the family of sets generated from B by adding the singleton set $1 = \{()\}$ and closing under binary products $X \times Y$ and set-theoretic function spaces Y^X. We shall consider some computability models whose family of datatypes is $\langle B\rangle$

First we may define a computability model $\mathsf{S}(B)$ with $|\mathsf{S}(B)| = \langle B\rangle$ (often called the **full set-theoretic model over** B) by letting $\mathsf{S}(B)[X, Y]$ consist of all set-theoretic functions $X \to Y$ for $X, Y \in \langle B\rangle$; that is, we consider all functions to be computable. However this model is of limited interest since it does not represent an interesting concept of computability

To do better we may start by noting that whatever the 'computable' functions between these sets are supposed to be, it is reasonable to expect that they will enjoy the following closure properties

1. For any $X \in \langle B\rangle$, the unique function $X \to 1$ is computable

2. For any $X, Y \in \langle B\rangle$, the projections $X \times Y \to X$, $X \times Y \to Y$ is computable

3. For any $X, Y \in \langle B \rangle$, the application function $Y^X \times X \to Y$ is computable

4. If $f : Z \to X$ and $g : Z \to Y$ is computable, so is their pairing $(f, g) : Z \to X \times Y$

5. If $f : X \to Y$ and $g : Y \to Z$ are computable, so is their composition $g \circ f : X \to Z$

6. If $f : Z \times X \to Y$ is computable, so is its transpose $\hat{f} : Z \to Y^X$

One possible approach is therefore to start by specifying some set C of functions between out datatypes that we wish to regard as "basic computable operations", and define a computability model $\mathsf{K}(B; C)$ over $\langle B \rangle$ whose operations are exactly the functions generated from C under the above closure conditions

Take $B = \{\mathbb{N}\}$; we shall often denote $\mathsf{S}(\{\mathbb{N}\})$ by S. Let C consist of the following basic operations: the zero function $\Lambda x.0 : \mathbb{N} \to 1$, the successor function $suc : \mathbb{N} \to \mathbb{N}$; and for each $X \in \langle B \rangle$, the primitive recursion operator $rec_X : (X \times X^{X \times \mathbb{N}} \times \mathbb{N}) \to X$ defined by

$$rec_X(x, f, 0) = 0$$
$$rec_X(x, f, n + 1) = f(rec_X(x, f, n), n)$$

the resulting model $\mathsf{K}(B; C)$ consists of exactly those operations of S definable in Gödel's **System T**

1.2.3 Weakly Cartesian Closed Models

Definition 1.5. Suppose **C** has weak products and a weak terminal. We say **C** is **weakly cartesian closed** if it is endowed with the following for each $A, B \in |\mathbf{C}|$:

- a choice of datatype $A \Rightarrow B \in |\mathbf{C}|$
- a partial function $\cdot_{AB} : (A \Rightarrow B) \times A \rightharpoonup B$, external to the structure of **C**

s.t. for any partial function $f : C \times A \rightharpoonup B$ the following are equivalent

1. f is represented by some $\overline{f} : \mathbb{C}[C \bowtie A, B]$, in the sense that if d represents (c, a) then $\overline{f}(d) \simeq f(c, a)$

2. f is represented by some total operation $\hat{f} : \mathbf{C}[C, A \Rightarrow B]$, in the sense that

$$\forall c \in C, a \in A \quad \hat{f}(c) \cdot_{AB} a \simeq f(c, a)$$

$\cdot_{AB}$ is represented by an operation $app_{AB} \in \mathbb{C}[(A \Rightarrow B) \bowtie A, B]$

Crucially, and in contrast to the definition of cartesian closed category, there is no requirement that f is unique. This highlights an important feature of our framework: in many models of interest, elements of $A \Rightarrow B$ will be **intensional** objects (programs or algorithms), and there may be many intensional objects giving rise to the same partial function $A \to B$

Example 1.4. Consider again the model of Example 1.1, comprising the partial Turing-computable functions $\mathbb{N} \rightharpoonup \mathbb{N}$. Here $\mathbb{N} \Rightarrow \mathbb{N}$ can only be $\mathbb{N}$, so we must provide a suitable operation $\cdot : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. This is done using the concept of a **universal Turing machine**. Let $T_0, T_1, ...$ be some sensibly chosen enumeration of all Turing machines for computing partial functions $\mathbb{N} \rightharpoonup \mathbb{N}$. Then there is a Turing machine that accepts two inputs e, a and returns the result of applying the machine T_e to the single input a. We may therefore take $\cdot$ to be the partial function computed by U

Clearly the partial functions $f : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$ representable within the model via the pairing operation from Example 1.1 are just the partial computable ones. We may also see that these coincide exactly with those represented by some total computable $\overline{f} : \mathbb{N} \to \mathbb{N}$, in the sense that $f(c, a) \simeq \tilde{f}(c) \cdot a$.

$\Leftarrow$: Given a computable $\tilde{f}$ the operation $\Lambda(c, a).\tilde{f}(c) \cdot a$ is clearly computable

$\Rightarrow$: s-m-n theorem

When endowed with this weakly cartisian closed structure, this computability model is known as **Kleene's first model** of K_1

Example 1.5. Now consider the model $\Lambda/\sim$; we shall write L for the set $\Lambda/\sim$ considered as the sole datatype in this model. Set $\mathsf{L} \Rightarrow \mathsf{L} = \mathsf{L} \bowtie \mathsf{L} = \mathsf{L}$. We may obtain a weakly cartesian closed structure by letting $\cdot$ be given by application. If $M \in \Lambda$ induces an operation in $[\mathsf{L} \bowtie \mathsf{L}, \mathsf{L}]$ representing some $f : \mathsf{L} \times \mathsf{L} \to \mathsf{L}$, then $\lambda x.\lambda y.M(pair\ x\ y)$ induces the corresponding operation in $[\mathsf{L}, \mathsf{L} \Rightarrow \mathsf{L}]$; conversely if N induces an operation in $[\mathsf{L}, \mathsf{L} \Rightarrow \mathsf{L}]$ then $\lambda z.N(fst\ z)(snd\ z)$ induces the corresponding one in $[\mathsf{L} \bowtie \mathsf{L}, \mathsf{L}]$

Example 1.6. For models of the form $\mathsf{K}(B; C)$, we naturally define $X \Rightarrow Y = Y^X$ and take $\cdot_{XY}$ to be ordinary function application. These models are

endowed with binary products, and it is immediate from closure condition 6 in Example 1.3 that they are weakly cartesian closed

Such models show that not every element of $X \Rightarrow Y$ need represent an operation in $\mathbf{C}[X, Y]$, or equivalently one in $\mathbf{C}[1, X \Rightarrow Y]$. This accords with the idea that our models consist of 'computable' operations acting on potentially 'non-computable' data: operations in $\mathbf{C}[X, Y]$ are computable, whereas elements of X need not be

1.2.4 Higher-Order Models

Definition 1.6. A **higher-order structure** is a computability model $\mathbf{C}$ possessing a weak terminal (I, i) and endowed with the following for each $A, B \in |\mathbf{C}|$

- a choice of datatype $A \Rightarrow B \in |\mathbf{C}|$
- a partial function $\cdot_{AB} : (A \Rightarrow B) \times A \rightharpoonup B$

We treat $\Rightarrow$ as right-associative and $\cdot$ as left-associative

The significance of the weak terminal (I, i) here is that it allows us to pick out a subset $A^\sharp$ of each $A \in |\mathbf{C}|$, namely the set of elements of the form $f(i)$ where $f \in \mathbf{C}[I, A]$ and $f(i) \downarrow$.

This is independent of the choice of (I, i): if $a = f(i)$ and (J, j) is another weak terminal, then composing f with $\Lambda x.i \in \mathbf{C}[J, I]$ gives $f' \in \mathbf{C}[J, A]$ with $f'(j) = a$.

Intuitively, we think of $A^\sharp$ as playing the role of the 'computable' elements of A, and i as some generic computable element. On the one hand, if $a \in A$ were computable, we would expect each $\Lambda x.a$ to be computable so that $a \in A^\sharp$; on the other hand, the image of a computable element under a computable operation should be computable, so that every element of $A^\sharp$ is computable.

Any weakly cartesian closed model $\mathbf{C}$ is a higher-structure.

Definition 1.7. A **higher-order (computability) model** is a higher-order structure $\mathbf{C}$ satisfying the following conditions for some (or equivalently any) weak terminal (I, i)

1. A partial function $f : A \rightharpoonup B$ is present in $\mathbf{C}[A, B]$ iff there exists $\hat{f} \in \mathbf{C}[I, A \Rightarrow B]$ s.t.

$$\hat{f}(i) \downarrow, \quad \forall a \in A. \hat{f}(i) \cdot a \simeq f(a)$$

2. For any $A, B \in |\mathbf{C}|$, there exists $k_{AB} \in (A \Rightarrow B \Rightarrow A)^\sharp$ s.t.

$$\forall a.k_{AB} \cdot a \downarrow, \quad \forall a, b.k_{AB} \cdot a \cdot b = a$$

3. For any $A, B, C \in |\mathbf{C}|$ there exits

$$s_{ABC} \in ((A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow C))^\sharp$$

s.t.

$$\forall f, g.s_{ABC} \cdot f \cdot g \downarrow, \quad \forall f, g, a.s_{ABC} \cdot f \cdot g \cdot a \simeq (f \cdot a) \cdot (g \cdot a)$$

The elements k and s correspond to combinators from combinatory logic. k allows us to construct **constant** maps in a computable way

A possible intuition for s is that it somehow does duty for an application operation $(B \Rightarrow C) \times B \rightharpoonup C$ within **C** itself, where the application may be performed uniformly in a parameter of type A.p

Proposition 1.8. *Suppose **C** is a higher-order model*

1. *for any $j < m$, there exists $\pi^m_j \in (A_0 \Rightarrow \cdots \Rightarrow A_{m-1} \Rightarrow A_j)^\sharp$ s.t.*

$$\forall a_0, \ldots, a_{m-1}.\pi^m_j \cdot a_0 \cdot \cdots \cdot a_{m-1} = a_j$$

2. *Suppose $m, n > 0$. Given*

$$f_j \in (A_0 \Rightarrow \cdots \Rightarrow A_{m-1} \Rightarrow B_j)^\sharp, \quad (j = 0, \ldots, n-1),$$
$$g \in (B_0 \Rightarrow \cdots \Rightarrow B_{n-1} \Rightarrow C)^\sharp$$

there exists $h \in (A_0 \Rightarrow \cdots \Rightarrow A_{m-1} \Rightarrow C)^\sharp$ s.t.

$$\forall a_0, \ldots, a_{m-1}.h \cdot a_0 \cdot \cdots \cdot a_{m-1} \simeq g \cdot (f_0 \cdot a_0 \cdot \cdots \cdot a_{m-1}) \cdot \cdots \cdot (f_{n-1} \cdot a_0 \cdot \cdots \cdot a_{m-1})$$

3. *Suppose $m > 0$. For any element $f \in (A_0 \Rightarrow \cdots \Rightarrow A_{m-1} \Rightarrow B)^\sharp$, there exists $f^\dagger \in (A_0 \Rightarrow \cdots \Rightarrow A_{m-1} \Rightarrow B)^\sharp$ s.t.*

$$\forall a_0, \ldots, a_{m-1}.f^\dagger \cdot a_0 \cdot \cdots \cdot a_{m-1} \simeq f \cdot a_0 \cdot \cdots \cdot a_{m-1}$$
$$\forall k < m. \forall a_0, \ldots, a_{k-1}.f^\dagger \cdot a_0 \cdot \cdots \cdot a_{k-1} \downarrow$$

$i_A = s_{A(A \Rightarrow A)A} \cdot k_{A \Rightarrow A} \cdot k_{AA} \in (A \Rightarrow A)^\sharp$

Proof. 1. consider

$T[x] \Rightarrow x$
$T[(E_1\ E_2)] \Rightarrow (T[E_1]\ T[E_2])$if x does not occur free in E
$T[\lambda x.E] \Rightarrow (\mathbf{K}\ T[E])$
$T[\lambda x.x] \Rightarrow \mathbf{I}$
$T[\lambda x.\lambda y.E] \Rightarrow T[\lambda x.T[\lambda y.E]]$if x occurs free in E
$T[\lambda x.(E_1\ E_2)] \Rightarrow (\mathbf{S}\ T[\lambda x.E_1]\ T[\lambda x.E_2])$if x occurs free in E_1 or E_2

so $A \Rightarrow B \Rightarrow B \to \lambda x^A y^B.y^B \to \mathbf{K}_{B \Rightarrow B,A} \cdot I_B$

□

If **C**, **D** are higher-order structures, we say **C** is a **full substructure** of **D** if

- $|\mathbf{C}| \subseteq |\mathbf{D}|$
- $\mathbf{C}[A, B] = \mathbf{D}[A, B]$ for all $A, B \in |\mathbf{C}|$
- some (or equivalently any) weak terminal in **C** is also a weak terminal in **D**
- the meaning of $A \Rightarrow B$ and $\cdot_{AB}$ in **C** and **D** coincide

Note that if (I, i) and (J, j) are weak terminals in **C** then $\Lambda x.j \in \mathbf{C}[I, J]$, so if (I, i) is a weak terminal in **D** then so is (J, j)

Theorem 1.9. *A higher-order structure is a higher-order model iff it is a full substructure of a weakly cartesian closed model*

Proof. Let **C** be a higher-order structure.

$\Leftarrow$: suppose **D** is weakly cartesian closed and **C** is a full substructure of **D** with a weak terminal (I, i)

1. For any $f \in \mathbf{C}[A, B] = \mathbf{D}[A, B]$ we have that $f \circ \pi_A \in \mathbf{D}[I \bowtie A, B]$ represents $\Lambda(x, a).f(a)$, which by definition 1.5 is in turn represented by some total $\hat{f} \in \mathbf{D}[I, A \Rightarrow B]$.

   Conversely, given $f : A \rightharpoonup B$ and $\hat{f} \in \mathbf{C}[I, A \Rightarrow B]$ with $\hat{f}(i) \downarrow$ and $\hat{f}(i) \cdot a \simeq f(a)$ for all a, take $\hat{g} = \hat{f} \circ (\Lambda x.i) \in \mathbf{C}[I, A \Rightarrow B] = \mathbf{D}[I, A \Rightarrow B]$ so that $\hat{g}$ is total and represents $g = \Lambda(x, a).f(a) : I \times A \rightharpoonup B$. Now let $\bar{g} \in \mathbf{D}[I \bowtie A, B]$ also represents g. Then $\bar{g} \circ \langle \Lambda a.i, \mathrm{id}_A \rangle \in \mathbf{D}[A, B] = \mathbf{C}[A, B]$ and it is routine to check that $\bar{g} \circ \langle \Lambda a.i, \mathrm{id}_A \rangle = f$

2. Suppose $A, B \in |\mathbf{C}|$. Let $k' \in \mathbf{D}[A, B \Rightarrow A]$ correspond to $\pi_A \in \mathbf{D}[A \bowtie B, A]$ as in definition 1.5, then $k'(a) \cdot b \simeq \pi_A(d)$. Let $\widehat{k'} \in \mathbf{D}[I, A \Rightarrow (B \Rightarrow A)]$ correspond to $k' \circ \pi'_A \in \mathbf{D}[I \bowtie A, B \Rightarrow A]$ where $\pi'_A \in \mathbf{D}[I \bowtie A, A]$ and take $k = \widehat{k'}(i)\ k \cdot a \cdot b = \widehat{k'}(i) \cdot a \cdot b = (k' \circ \pi'_A(i, a)) \cdot b = k'(a) \cdot b = a$

3.

$\Rightarrow$: Suppose $\mathbf{C}$ is a higher-order model, with (I, i) a weak terminal. We build a weakly cartesian closed model $\mathbf{C}^\times$ into which $\mathbf{C}$ embeds fully as follows:

- Datatypes of $\mathbf{C}^\times$ are sets $A_0 \times \cdots \times A_{m-1}$, where $m > 0$ and $A_0, \ldots, A_{m-1} \in |\mathbf{C}|$
- If $D = A_0 \times \cdots \times A_{m-1}$ and $E = B_0 \times \cdots \times B_{n-1}$ where $m, n > 0$ the operations in $\mathbf{C}^\times[D, E]$ are those partial functions $f : D \rightharpoonup E$ of the form

  $$f = \Lambda(a_0, \ldots, a_{m-1}).(f_0 \cdot a_0 \cdot \cdots \cdot a_{m-1}, \ldots, f_{n-1} \cdot a_0 \cdot \cdots \cdot a_{m-1})$$

  where $f_j \in (A_0 \Rightarrow \cdots \Rightarrow A_{m-1} \Rightarrow B_j)^\sharp$ for each j; we say that $f_0, \ldots, f_{n-1}$ **witness** the operation f. Note that for $(f_0 \cdot a_0 \cdot \cdots \cdot a_{m-1}, \ldots, f_{n-1} \cdot a_0 \cdot \cdots \cdot a_{m-1})$ to be defined, it is necessary that all its components be defined

It remains to check the relevant properties of $\mathbf{C}^\times$. That $\mathbf{C}^\times$ is a computability model is straightforward: the existence of identities follows from part 1 of Proposition 1.8 and composition from part 2. $\mathbf{C}^\times$ has standard products and that (I, i) is a weak terminal in $\mathbf{C}^\times$.

Now let's show that $\mathbf{C}^\times$ is weakly cartesian closed. Given $D = A_0 \times \cdots \times A_{m-1}$ and $E = B_0 \times \cdots \times B_{n-1}$ with $m, n > 0$, take $C_j = A_0 \Rightarrow \cdots \Rightarrow A_{m-1} \Rightarrow B_j$ for each j, and let $D \Rightarrow E$ be the set of tuples $(f_0, \ldots, f_{n-1}) \in C_0 \times \cdots \times C_{n-1}$ witnessing operations in $\mathbf{C}^\times[D, E]$. The application $\cdot_{DE}$ is then given by

$$(f_0, \ldots, f_{n-1}) \cdot_{DE} (a_0, \ldots, a_{m-1}) \simeq (f_0 \cdot a_0 \cdot \cdots \cdot a_{m-1}, \ldots, f_{n-1} \cdot a_0 \cdot \cdots \cdot a_{m-1})$$

Next, given an operation $g \in \mathbf{C}^\times[G \times D, E]$ witnessed by operations $g_0, \ldots, g_{n-1}$ in $\mathbf{C}$, take $g_0^\dagger, \ldots, g_{n-1}^\dagger$ as in Proposition 1.8 (3); then $g_0^\dagger, \ldots, g_{n-1}^\dagger$ witness the corresponding total operation $\hat{g} \in \mathbf{C}^\times[G, D \Rightarrow E]$. Conversely, the witnesses for any such total $\hat{g}$ also witness the corresponding g $\square$

1.2.5 Typed Partial Combinatory Algebras

The following definition captures roughly what is left of a higher-order model once the operations are discarded

Definition 1.10. 1. A **partial applicative structure A** consists of

- an inhabited family $|\mathbf{A}|$ of datatypes $A, B, ...$ (indexed by some set T)
- a (right-associative) binary operation $\Rightarrow$ on $|\mathbf{A}|$
- for each $A, B \in |\mathbf{A}|$, a partial function $\cdot_{AB} : (A \Rightarrow B) \times A \rightharpoonup B$

2. A **typed partial combinatory algebra** (TPCA) is a partial applicative structure **A** satisfying the following conditions

   (a) For any $A, B \in |\mathbf{A}|$, there exists $k_{AB} \in A \Rightarrow B \Rightarrow A$ s.t.

   $$\forall a.k \cdot a \downarrow, \quad \forall a, b.k \cdot a \cdot b = a$$

   (b) For any $A, B, C \in |\mathbf{A}|$, there exists $s_{ABC} \in (A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow C)$ s.t.

   $$\forall f, g.s \cdot f \cdot g \downarrow, \quad \forall f, g, a.s \cdot f \cdot g \cdot a \simeq (f \cdot a) \cdot (g \cdot a)$$

   A TPCA is **total** if all the application operations $\cdot_{AB}$ are total

Any higher-order model yields an underlying TPCA. However, in passing to this TPCA we lose the information that says which element of $A \Rightarrow B$ are supposed to represent operations.

Definition 1.11. 1. If $\mathbf{A}^\circ$ denotes a partial applicative structure, a **partial applicative substructure** $\mathbf{A}^\sharp$ of $\mathbf{A}^\circ$ consists of a subset $A^\sharp \subseteq A$ for each $A \in |\mathbf{A}^\circ|$ s.t.

   - if $f \in (A \Rightarrow B)^\sharp$, $a \in A^\sharp$ and $f \cdot a \downarrow$ in $\mathbf{A}^\circ$, then $f \cdot a \in B^\sharp$

   such a pair $(\mathbf{A}^\circ; \mathbf{A}^\sharp)$ is called a **relative partial applicative structure**

2. A **relative TPCA** is a relative partial applicative structure $(\mathbf{A}^\circ, \mathbf{A}^\sharp)$ s.t. there exist elements k_{AB}, s_{ABC} in $\mathbf{A}^\sharp$ witnessing that $\mathbf{A}^\circ$ is a TPCA

A relative TPCA $(\mathbf{A}^\circ, \mathbf{A}^\sharp)$ is **full** if $\mathbf{A}^\sharp = \mathbf{A}^\circ$. We will use **A** to range over both ordinary TPCAs and relative ones (writing $\mathbf{A}^\circ$, $\mathbf{A}^\sharp$ for the two components of **A** in the latter case), so that in effect we identify an ordinary

TPCA $\mathbf{A}$ with the relative TPCA $(\mathbf{A}; \mathbf{A})$. Indeed, we may sometimes refer to ordinary TPCAs as 'full TPCAs' . Clearly the models K_1 and $\Lambda/\sim$ are full, while in general $\mathsf{K}(B; C)$ is not: rather, it is a relative TPCA $\mathbf{A}$ in which $\mathbf{A}^\circ$ is a full set-theoretic type structure whilst $\mathbf{A}^\sharp$ consists of only the C-computable elements

Theorem 1.12. *There is a canonical bijection between higher-order models and relative TPCAs*

Proof. First suppose $\mathbf{C}$ is a higher-order model, and let $\mathbf{A}^\circ$ be its underlying partial applicative structure. Take (I, i) a weak terminal in $\mathbf{C}$, and for any $A \in |\mathbf{C}|$, define $A^\sharp = \{g(i) \mid g \in \mathbf{C}[I, A], g(i)\downarrow\}$. As noted there, this is independent of the choice of (I, i); in fact, it is easy to see that $a \in A^\sharp$ iff (A, a) is a weak terminal. To see that the $A^\sharp$ form an applicative substructure, suppose $f \in (A \Rightarrow B)^\sharp$ is witnessed by $f' \in \mathbf{C}[I, A \Rightarrow B]$ and $a \in A^\sharp$ is witnessed by $a' \in \mathbf{C}[I, A]$, and suppose further that $f \cdot a = b$. Take $\check{f'} \in \mathbf{C}[A \Rightarrow B]$ corresponding to f'; then $\check{f'}(a) = b$ and so $\check{f'} \circ a'$ witnesses $b \in B^\sharp$

Let $\mathbf{A}^\sharp$ denote the substructure formed by the sets $A^\sharp$. It is directly build into Definition **??** that there are elements k_{AB}, s_{ABC} in $\mathbf{A}^\sharp$ with the properties required by Definition 1.17; thus $(\mathbf{A}^\circ; \mathbf{A}^\sharp)$ is a relative TPCA

For the converse, suppose $\mathbf{A}$ is a relative TPCA. Take $|\mathbf{C}| = |\mathbf{A}^\circ|$ and for $A, B \in |\mathbf{C}|$, let $\mathbf{C}[A, B]$ consist of all partial functions $\Lambda a.f \cdot a$ for $f \in (A \Rightarrow B)^\sharp$. To see that $\mathbf{C}$ has identities, for any $A \in |\mathbf{C}|$, we have

$$i_A = s_{A(A\Rightarrow A)A} \cdot k_{A(A\Rightarrow A)} \cdot k_{AA} \in (A \Rightarrow A)^\sharp$$

and clearly i_A induces $\mathrm{id}_A \in \mathbf{C}[A, A]$. For composition, given operations $f \in \mathbf{C}[A, B]$, $g \in \mathbf{C}[B, C]$ induced by $f' \in (A \Rightarrow B)^\sharp$, $g' \in (B \Rightarrow C)^\sharp$, we have that $g \circ f \in \mathbf{C}[A, C]$ is induced by $s_{ABC} \cdot (k_{(B\Rightarrow C)} \cdot g) \cdot f$. Thus $\mathbf{C}$ is a computability mode

For a weak terminal, take any $U \in |\mathbf{C}|$ and let $I = U \Rightarrow U$ and $i = i_U$ as defined above. Then for any A we have that $k_{IA} \cdot i \in (A \Rightarrow U \Rightarrow U)^\sharp$ induces $\Lambda a.i \in \mathbf{C}[A, I]$

To turn $\mathbf{C}$ into a higher-order structure, we take $\Rightarrow$ and $\cdot$ as in $\mathbf{A}^\circ$. We may now verify that for any A we have

$$A^\sharp = \{g(i) \mid g \in \mathbf{C}[I, A], g(i)\downarrow\}$$

so that the present meaning of $A^\sharp$ coincides with its meaning in Section **??**. For given $a \in A^\sharp$ we have $k_{AI} \cdot a \in (I \Rightarrow A)^\sharp$ inducing an operation g with $g(i) = a$. Conversely, given $g \in \mathbf{C}[I, A]$ with $g(i)\downarrow$ we have that $g(i) = g' \cdot i$

for some $g' \in (I \Rightarrow A)^\sharp$ (by definition, g is of the form $\Lambda a.f \cdot a$); but $i \in I^\sharp$ so $g(i) \in A^\sharp$

By applying the above equation to the type $A \Rightarrow B$, we see that conditions 1 and 2 of Definition 1.7 are satisfied, and conditions 3 and 4 are immediate from the k, s conditions in Definition 1.17. Thus $\mathbf{C}$ is a higher-order model $\square$

In the setting of a relative TPCA $\mathbf{A}$, we have a natural **degree structure** on the elements of $\mathbf{A}^\circ$. Specifically, if $a \in A$ and $b \in B$ where $A, B \in |\mathbf{A}^\circ|$, let us write $a \gg b$ if there exists $f \in \mathbf{A}^\sharp(A \Rightarrow B)$ with $f \cdot a = b$

If $|\mathbf{A}|$ consists of just a single datatype, then TPCA is just a single set A equipped with a partial 'application' operation $\cdot : A \times A \rightharpoonup A$ s.t. for some $k, s \in A$ we have

$$\forall x, y.k \cdot x \cdot y = x, \quad \forall x, y.s \cdot x \cdot y \downarrow, \quad \forall x, y, z.s \cdot x \cdot y \cdot z \simeq (x \cdot z) \cdot (y \cdot z)$$

We call such a structure an **partial combinatory algebra** (or PCA)

1.2.6 Lax Models

For simplicity, we have worked so far with a simple definition of computability model in which operations are required to be closed under ordinary composition of partial functions. It turns out, however, that with a few refinements, practically all the general theory presented in this chapter goes through under a somewhat milder assumption.

Definition 1.13. A **lax computability model $\mathbf{C}$** over a set T of **type names** consists of

- an indexed family $|\mathbf{C}| = \{\mathbf{C}(\tau) \mid \tau \in \mathsf{T}\}$ of sets, called the **datatypes** of $\mathbf{C}$
- for each $\sigma, \tau \in \mathsf{T}$, a set $\mathbf{C}[\sigma, \tau]$ of partial functions $f : \mathbf{C}(\sigma) \to \mathbf{C}(\tau)$, called the **operations** of $\mathbf{C}$

s.t.

1. for each $\tau \in \mathsf{T}$, the identity function $\mathrm{id} : \mathbf{C}(\tau) \to \mathbf{C}(\tau)$ is in $\mathbf{C}(\tau, \tau)$
2. for any $f \in \mathbf{C}[\rho, \sigma]$ and $g \in \mathbf{C}[\sigma, \tau]$, there exists $h \in \mathbf{C}[\rho, \tau]$ with $h(a) \geq g(f(a))$ for all $a \in \mathbf{C}(\rho)$

We may refer to h here as a **supercomposition** of f and g.

We sometimes refer to our standard computability models as **strict** when we wish to emphasize the contrast with lax models. Of course, for total computability models, the distinction evaporates completely.

One possible motivation for the concept of lax model is that it is often natural to think of an application $f(a)$ in terms of some computational agent F representing f being placed 'alongside' a representation A of a to yield a composite system $F \mid A$, which may then evolve in certain ways via interactions between F and A. If an agent G representing g is then placed alongside this to yield a system $G \mid F \mid A$, there is the possibility that G may interact 'directly' with F rather than just with the result obtained from $F \mid A$; thus, $G \mid F$ might admit other behaviours not accounted for by $g \circ f$. (For a precise example of this in process algebra, see Longley [183].)

The notion of a **(relative) lax TPCA** is given by replacing the axioms for s_{ABC} in Definition 1.17 with

$$\forall f, g.s \cdot f \cdot g \downarrow, \quad \forall f, g, a.s \cdot f \cdot g \cdot a \succeq (f \cdot a) \cdot (g \cdot a)$$

The definitions of weak products and weak terminal may be carried over unchanged to the setting of lax computability models; note that $\langle f, g \rangle$ is still required to be a pairing in the 'strict' sense that its domain coincides precisely with $\mathrm{dom}\, f \cap \mathrm{dom}\, g$. The definition of weakly cartesian closed model is likewise unchanged, although one should note that in the lax setting, whether a given model is weakly cartesian closed may be sensitive to the choice of the type operator $\bowtie$.

For the definition of a lax higher-order model, we simply replace '$\simeq$' by '$\succeq$' in condition 4(?) of Definition 1.7

Theorem 1.14. *1. Any lax higher-order model is a full substructure of a lax weakly cartesian closed model*

*2. If **D** is a lax weakly cartesian closed model in which some weak terminal (I, i) is a weak unit, any full substructure of **D** containing I is a lax higher-order model*

1.2.7 Type worlds

Definition 1.15. 1. A **type world** is simply a set T of **type names** σ, optionally endowed with any or all of the following:

(a) a **fixing map**, assigning a set $\mathsf{T}[\sigma]$ to certain type names $\sigma \in \mathsf{T}$

(b) a **product structure**, consisting of a total binary operation $(\sigma, \tau) \mapsto \sigma \times \tau$

(c) an **arrow structure**, consisting of a total binary operation $(\sigma, \tau) \mapsto \sigma \to \tau$

2. A **computability model over** a type world T is a computability model $\mathbf{C}$ with index set T (so that $|\mathbf{C}| = \{\mathbf{C}(\sigma) \mid \sigma \in \mathsf{T}\}$) subject to the following conventions

   (a) If T has a fixing map, then $\mathbf{C}(\sigma) = T[\sigma]$ whenever $\mathsf{T}(\sigma)$ is defined

   (b) If T has a product structure, then $\mathbf{C}$ has weak products and for any $\sigma, \tau \in \mathsf{T}$ we have $\mathbf{C}(\sigma \times \tau) = \mathbf{C}(\sigma) \bowtie \mathbf{C}(\tau)$

   (c) If T has an arrow structure, then $\mathbf{C}$ is a higher-order model and for any $\sigma, \tau \in \mathsf{T}$ we have $\mathbf{C}(\sigma \to \tau) = \mathbf{C}(\sigma) \Rightarrow \mathbf{C}(\tau)$

   (d) If T has both a product and an arrow structure, then $\mathbf{C}$ is weakly cartesian closed

Example 1.7. The one-element type world $\mathsf{O} = \{*\}$ with just the arrow structure $* \to * = *$. TPCAs over this type world are precisely (untyped) PCAs; both K_1 and $\Lambda/\sim$ are examples

Example 1.8. If $\beta_0, \ldots, \beta_{n-1}$ are distinct **basic type names** and $B_0, \ldots, B_{n-1}$ are sets, we may define the type word $\mathsf{T}^{\to}(\beta_0 = B_0, \ldots, \beta_{n-1} = B_{n-1})$ to consist of formal type expressions freely constructed from $\beta_0, \ldots, \beta_{n-1}$ via $\to$, fixing the interpretation of each β_i at B_i. This type world has a fixing map and an arrow structure, but no product. We may write just $\mathsf{T}^{\to}(\beta_0, \ldots, \beta_{n-1})$ if we do not wish to constrain the interpretation of the β_i

A typical example is the type world $\mathsf{T}^{\to}(\mathtt{N} = \mathbb{N})$. Models over this type would correspond to **finite type structures** over $\mathbb{N}$; the models $\mathsf{K}(B; C)$ are examples

Type world $\mathsf{T}^{\to}(\mathtt{N} = \mathbb{N}_\perp)$ where $\mathbb{N}_\perp$ is the set of natural numbers together with an additional element $\perp$ representing 'non-termination'. Whereas $\mathbb{N}$ may be used to model actual **results** of computation, we may think of $\mathbb{N}_\perp$ as representing some computational **process** which may or may not return a natural number.

Example 1.9. Similarly, we define $\mathsf{T}^{\to\times} = (\beta_0 = B_0, \ldots, \beta_{n-1} = B_{n-1})$ to be the type world consisting of type expressions freely constructed form $\beta_0, \ldots, \beta_{n-1}$ via $\to$ and $\times$, fixing the interpretation of each β_i at B_i. If no fixing map is required, we write just $\mathsf{T}^{\to\times}(\beta_0, \ldots, \beta_{n-1})$

Type worlds featuring a **unit type** (denoted by $\mathtt{1}$) are also useful. We shall write $\mathsf{T}^{\to\times\mathtt{1}}(\beta_0 = B_0, \ldots, \beta_{n-1} = B_{n-1})$ for the type world

$$\mathsf{T}^{\to\times}(\mathtt{1} = \{()\}, \beta_0 = B_0, \ldots, \beta_{n-1} = B_{n-1})$$

We will often refer to the type names in a type world simply as **types**, and use ρ, σ, τ to range over them. When dealing with formal type expressions, we adopt the usual convention that $\to$ is right-associative, so that $\rho \to \sigma \to \tau$ means $\rho \to (\sigma \to \tau)$. For definiteness, we may also declare that $\times$ is right-associative, although in practice we shall not always bother to distinguish between $(\rho \times \sigma) \times \tau$ and $\rho \times (\sigma \times \tau)$. We consider $\times$ as binding more tightly than $\to$

We shall use the notation $\sigma_0, \ldots, \sigma_{r-1} \to \tau$ as an abbreviation for $\sigma_0 \to \sigma_1 \to \ldots \to \sigma_{r-1} \to \tau$ (allowing this to mean τ in the sense $r = 0$). This allows us to express our intention regarding which objects are to be thought of as 'arguments' to a given operation: for instance, the types $\mathtt{N}, \mathtt{N}, \mathtt{N} \to \mathtt{N}$ and $\mathtt{N}, \mathtt{N} \to (\mathtt{N} \to \mathtt{N})$ are formally the same, but in the first case we are thinking of a three-argument operation returning a natural number, while in the second we are thinking of a two-argument operation returning a function $\mathbb{N} \to \mathbb{N}$. We also write $\sigma^{(r)} \to \tau$ for the type $\sigma, \ldots, \sigma \to \tau$ with r arguments. The notation σ^r is reserved for the r-fold product type $\sigma \times \cdots \times \sigma$

Proposition 1.16. *Any type $\sigma \in \mathsf{T}^{\to}(\beta_0, \ldots, \beta_{n-1})$ may be uniquely written in the form $\sigma_0, \ldots, \sigma_{r-1} \to \beta_i$*

We shall call this the **argument form** of σ. The importance of this is that it provides a useful induction principle for types: if a property holds for $\sigma_0, \ldots, \sigma_{r-1} \to \beta_i$ whenever it holds for each of $\sigma_0, \ldots, \sigma_{r-1}$, then it holds for all $\sigma \in \mathsf{T}^{\to}(\beta_0, \ldots, \beta_{n-1})$. We shall refer to this as **argument induction**; it is often preferable as an alternative to the usual **structural induction** on types

Closely associated with argument form is the notion of the **level** of a type σ: informally, the stage at which σ appears in the generation of $\mathsf{T}^{\to}(\beta_0, \ldots, \beta_{n-1})$ via argument induction:

$$\mathrm{lv}(\beta_i) = 0$$
$$\mathrm{lv}(\sigma_0, \ldots, \sigma_{r-1} \to \beta_i) = 1 + \max_{i<r} \mathrm{lv}(\sigma_i) \quad (r \geq 1)$$

When working with $\mathsf{T}^{\to\times}(\beta_0, \ldots, \beta_{n-1})$, it is natural to augment this definition with

$$\mathrm{lv}(\sigma \times \tau) = \max(\mathrm{lv}(\sigma), \mathrm{lv}(\tau))$$

We may define the **pure type of level k over σ**, written $\bar{k}[\sigma]$:

$$\bar{0}[\sigma] = \sigma, \quad \overline{k+1}[\sigma] = \bar{k}[\sigma] \to \sigma$$

For type worlds generated by a single base type β, we may write simply $\bar{k}$ for $\bar{k}[\beta]$. For instance, in the type word $\mathsf{T}^{\to}(\mathbb{N})$ we write $\bar{2}$ for the type $(\mathtt{N} \to \mathtt{N}) \to \mathtt{N}$.

1.3 Computational Structure in Higher-Order Models

1.3.1 Combinatory Completeness

Combinatory completeness can be seen as a syntactic counterpart to the notion of weakly cartesian closed model. In essence, combinatory completeness asserts that any operation definable by means of a formal expression over A (constructed using application) is representable by an element of A itself.

Definition 1.17. 1. A **partial applicative structure A** consists of

- an inhabited family $|\mathbf{A}|$ of datatypes $A, B, ...$ (indexed by some set T)
- a (right-associative) binary operation $\Rightarrow$ on $|\mathbf{A}|$
- for each $A, B \in |\mathbf{A}|$, a partial function $\cdot_{AB} : (A \Rightarrow B) \times A \rightharpoonup B$

2. A **typed partial combinatory algebra** (TPCA) is a partial applicative structure **A** satisfying the following conditions

   (a) For any $A, B \in |\mathbf{A}|$, there exists $k_{AB} \in A \Rightarrow B \Rightarrow A$ s.t.

   $$\forall a.k \cdot a \downarrow, \quad \forall a, b.k \cdot a \cdot b = a$$

   (b) For any $A, B, C \in |\mathbf{A}|$, there exists $s_{ABC} \in (A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow C)$ s.t.

   $$\forall f, g.s \cdot f \cdot g \downarrow, \quad \forall f, g, a.s \cdot f \cdot g \cdot a \simeq (f \cdot a) \cdot (g \cdot a)$$

   A **lax TPCA** is obtained from a TPCA change '$\simeq$' to '$\succeq$' in the axiom s

3. If $\mathbf{A}^\circ$ denotes a partial applicative structure, a **partial applicative substructure** $\mathbf{A}^\sharp$ of $\mathbf{A}^\circ$ consists of a subset $A^\sharp \subseteq A$ for each $A \in |\mathbf{A}^\circ|$ s.t.

   - if $f \in (A \Rightarrow B)^\sharp$, $a \in A^\sharp$ and $f \cdot a \downarrow$ in $\mathbf{A}^\circ$, then $f \cdot a \in B^\sharp$

   such a pair $(\mathbf{A}^\circ; \mathbf{A}^\sharp)$ is called a **relative partial applicative structure**

4. A **relative TPCA** is a relative partial applicative structure $(\mathbf{A}^\circ, \mathbf{A}^\sharp)$ s.t. there exist elements k_{AB}, s_{ABC} in $\mathbf{A}^\sharp$ witnessing that $\mathbf{A}^\circ$ is a TPCA

Definition 1.18. Suppose **A** is a relative partial applicative structure over T

1. The set of well-typed **applicative expressions** $e : \sigma$ over $\mathbf{A}$ is defined inductively as follows

   - for each $\sigma \in \mathsf{T}$, we have an unlimited supply of variables $x^\sigma : \sigma$
   - for each $\sigma \in \mathsf{T}$ and $a \in \mathbf{A}^\sharp(\sigma)$, we have a **constant** symbol $c_a : \sigma$ (we shall often write c_a simply as a)
   - If $e : \sigma \to \tau$ and $e' : \sigma$ are applicative expressions, then ee' is an applicative expression of type τ.

   We write $V(e)$ for the set of variables appearing in e

2. A **valuation** in $\mathbf{A}$ is a function v assigning to certain variables x^σ an element $v(x^\sigma) \in \mathbf{A}^\circ(\sigma)$. Given an applicative expression e and a valuation v covering $V(e)$, the value $[\![e]\!]_v$, when defined, is given inductively by
   $$[\![x^\sigma]\!]_v = v(x), \quad [\![c_a]\!]_v = a, \quad [\![ee']\!]_\nu \simeq [\![e]\!]_v \cdot [\![e']\!]_v$$
   Note that if $e : \tau$ and $[\![e]\!]_v$ is defined then $[\![e]\!]_v \in \mathbf{A}^\circ(\tau)$.

Note that for any v with $\mathrm{ran}(v) \in \mathbf{A}^\sharp$, we can prove $[\![e : \tau]\!]_v \in \mathbf{A}^\sharp(\tau)$ by induction:

1. If e is of the form x^τ
2. If e is of the form c_a where $a \in \mathbf{A}^\sharp(\tau)$
3. If e is of the form $e'e''$ where $e' : \sigma \to \tau$ and $e'' : \sigma$.

   $[\![e]\!]_v = [\![e']\!]_v \cdot [\![e'']\!]_v$ where $[\![e']\!]_v \in \mathbf{A}^\sharp(\sigma \to \tau)$ and $[\![e'']\!]_v \in \mathbf{A}^\sharp(\sigma)$. Since $\mathbf{A}^\sharp$ is a substructure of $\mathbf{A}^\circ$, if $[\![e']\!]_v \cdot [\![e'']\!]_v \downarrow$, then $[\![e]\!] \in \mathbf{A}^\sharp(\tau)$

Definition 1.19. Let $\mathbf{A}$ be a relative partial applicative structure. We say $\mathbf{A}$ is **lax combinatory complete** if for every applicative expression $e : \tau$ over $\mathbf{A}$ and every variable x^σ, there is an applicative expression $\lambda^* x^\sigma.e$ with $V(\lambda^* x^\sigma.e) = V(e) - \{x^\sigma\}$ s.t. for any valuation v covering $V(\lambda^* x^\sigma.e)$ and any $a \in \mathbf{A}^\circ(\sigma)$ we have

$$[\![\lambda^* x^\sigma.e]\!]_v \downarrow, \quad [\![\lambda^* x^\sigma.e]\!]_v \cdot a \succeq [\![e]\!]_{v, x \mapsto a}$$

We say $\mathbf{A}$ is **strictly combinatory complete** if this holds with '$\simeq$' in place of '$\succeq$'

Theorem 1.20. *A (relative) partial applicative structure $\mathbf{A}$ is a lax (relative) TPCA iff it is lax combinatory complete*

Proof. If **A** is lax combinatory complete, then for any ρ, σ, τ we may define

$$k_{\sigma\tau} = [\![\lambda^* x^\sigma.(\lambda^* y^\tau.x)]\!]_\emptyset$$
$$s_{\rho\sigma\tau} = [\![\lambda^* x^{\rho\to\sigma\to\tau}.(\lambda^* y^{\rho\to\sigma}.(\lambda^* z^\rho.xz(yz)))]\!]_\emptyset$$

Conversely, if **A** is a lax TPCA, then given any suitable choice of elements k and s for **A**, we may define $\lambda^* x^\sigma.e$ by induction on the structure of e:

$$\begin{aligned}
\lambda^* x^\sigma.x &= s_{\sigma(\sigma\to\sigma)} k_{\sigma(\sigma\to\sigma)} k_{\sigma\sigma} \\
\lambda^* x^\sigma.a &= k_{\tau\sigma} a && \text{for each } a \in \mathbf{A}^\sharp(\tau) \\
\lambda^* x^\sigma.ee' &= s_{\sigma\tau\tau'}(\lambda^* x^\sigma.e)(\lambda^* x^\sigma.e') && \text{if } e : \tau \to \tau', e' : \tau \text{ and } ee' \text{ contains } x
\end{aligned}$$

□

The same argument shows that **A** is a strict TPCA iff it is strictly combinatory complete

we often tacitly suppose that a TPCA **A** comes equipped with some choice of k and s drawn from A♯, and in this case we shall use the notation $\lambda^* x.e$ for the applicative expression given by the above proof. Since all the constants appearing in e are drawn from $A^\sharp$, the same will be true for $\lambda^* x.e$.

In TPCAs constructed as syntactic models for untyped or typed λ-calculi (as in Example 3.1.6 or Section 3.2.3), the value of $\lambda^* x.e$ coincides with $\lambda x.e$. However, the notational distinction is worth retaining, since the term $\lambda^* x.e$ as defined above is not syntactically identical to $\lambda x.e$.

More generally, we may consider terms of the λ-calculus as **meta-expressions** for applicative expressions. Specifically any such λ-term M can be regarded as denoting an applicative expression $M^\dagger$ as follows:

$$x^\dagger = x, \quad c_a^\dagger = c_a, \quad (MN)^\dagger = M^\dagger N^\dagger, \quad (\lambda x.M)^\dagger = \lambda^* x.(M^\dagger)$$

Some caution is needed here, however, because β-equivalent meta-expressions do not always have the same meaning

Example 1.10. Consider the two meta-expressions $(\lambda x.(\lambda y.y)x)$ and $\lambda x.x$. Although these are β-equivalent, the first expands to $s(ki)i$ and the second to i, where $i \equiv skk$.

The moral here is that β-reductions are not valid underneath λ^*-abstractions: in this case, the reduction $(\lambda^* y.y)x \twoheadrightarrow x$ is not valid underneath λ^*. However at least for the definition of λ^* given above, β-reductions at top level are valid.

Proposition 1.21. *1. If M is a meta-expression, x is a variable and a is a constant or variable, then $[\![((\lambda x.M)a)^\dagger]\!]_v \succeq [\![M[x \mapsto a]^\dagger]\!]$*

2. If M, N are meta-expressions, $x \notin FV(N)$, no free occurrence of x in M occurs under a λ, and $[\![N^\dagger]\!]_v \downarrow$, then $[\![((\lambda x.M)N)^\dagger]\!]_v \succeq [\![M[x \mapsto N]^\dagger]\!]_v$

Proof. Longley's PhD thesis □

From now on, we will not need to distinguish formally between meta-expressions and the applicative expressions they denote. For the remainder of this chapter we shall use the λ^* notation for such (meta-)expressions, retaining the asterisk as a reminder that the usual rules of λ-calculus are not always valid.

1.3.2 Pairing

Definition 1.22. 1. A **type world** is simply a set T of **type names** σ, optionally endowed with any or all of the following:

(a) a **fixing map**, assigning a set $\mathsf{T}[\sigma]$ to certain type names $\sigma \in \mathsf{T}$

(b) a **product structure**, consisting of a total binary operation $(\sigma, \tau) \mapsto \sigma \times \tau$

(c) an **arrow structure**, consisting of a total binary operation $(\sigma, \tau) \mapsto \sigma \to \tau$

2. A **computability model over** a type world T is a computability model $\mathbf{C}$ with index set T (so that $|\mathbf{C}| = \{\mathbf{C}(\sigma) \mid \sigma \in \mathsf{T}\}$) subject to the following conventions

(a) If T has a fixing map, then $\mathbf{C}(\sigma) = T[\sigma]$ whenever $\mathsf{T}(\sigma)$ is defined

(b) If T has a product structure, then $\mathbf{C}$ has weak products and for any $\sigma, \tau \in \mathsf{T}$ we have $\mathbf{C}(\sigma \times \tau) = \mathbf{C}(\sigma) \bowtie \mathbf{C}(\tau)$

(c) If T has an arrow structure, then $\mathbf{C}$ is a higher-order model and for any $\sigma, \tau \in \mathsf{T}$ we have $\mathbf{C}(\sigma \to \tau) = \mathbf{C}(\sigma) \Rightarrow \mathbf{C}(\tau)$

(d) If T has both a product and an arrow structure, then $\mathbf{C}$ is weakly cartesian closed

Theorem 1.23. *There is a canonical bijection between higher-order models and relative TPCAs*

Let $\mathbf{A}$ be a relative TPCA (which is combinatory complete) over a type world T with arrow structure, and suppose that $\mathbf{A}$ (considered as a higher-order model) has weak products, inducing a product structure $\times$ on T. This means that for any $\sigma, \tau \in \mathsf{T}$ there are elements

$$fst \in \mathbf{A}^\sharp((\sigma \times \tau) \to \sigma), \quad snd \in \mathbf{A}^\sharp((\sigma \times \tau) \to \tau)$$

And for each $\sigma, \tau \in \mathsf{T}$ a **paring** operation

$$pair \in \mathbf{A}^\sharp(\sigma \to \tau \to (\sigma \times \tau))$$

s.t.

$$\forall a \in \mathbf{A}^\circ(\sigma), b \in \mathbf{A}^\circ(\tau).\ fst \cdot (pair \cdot a \cdot b) = a \wedge snd \cdot (pair \cdot a \cdot b) = b$$

Proposition 1.24. *A higher-order model with weak products has pairing iff it is weakly cartesian closed*

Lemma 1.25 (1.8)**.** *Suppose $m, n > 0$. Given*

$$f_j \in (A_0 \Rightarrow \cdots \Rightarrow A_{m-1} \Rightarrow B_j)^\sharp, \quad (j = 0, \ldots, n-1),$$
$$g \in (B_0 \Rightarrow \cdots \Rightarrow B_{n-1} \Rightarrow C)^\sharp$$

there exists $h \in (A_0 \Rightarrow \cdots \Rightarrow A_{m-1} \Rightarrow C)^\sharp$ s.t.

$$\forall a_0, \ldots, a_{m-1}. h \cdot a_0 \cdot \cdots \cdot a_{m-1} \simeq g \cdot (f_0 \cdot a_0 \cdot \cdots \cdot a_{m-1}) \cdot \cdots \cdot (f_{n-1} \cdot a_0 \cdot \cdots \cdot a_{m-1})$$

Proof. The binary partial functions representable in $\mathbf{A}^\sharp((\rho \times \sigma) \to \tau)$ are exactly those representable in $\mathbf{A}^\sharp(\rho \to \sigma \to \tau)$

Given $f \in \mathbf{A}^\sharp((\rho \times \sigma) \to \tau)$, by Proposition 1.8, we have $h \in \mathbf{A}^\sharp(\rho \to \sigma \to \tau)$ where

$$\forall a, b.\ h \cdot a \cdot b \simeq f \cdot (pair\ \cdot a \cdot b)$$

Given $f \in \mathbf{A}^\sharp(\rho \to \sigma \to \tau)$, by the same Proposition, we have $h \in \mathbf{A}^\sharp((\rho \times \sigma) \to \tau)$ where

$$\forall a, b.\ h \cdot c \simeq f \cdot (fst \cdot c) \cdot (snd \cdot c)$$

□

Henceforth we shall generally work with pair in preference to the 'external' pairing of operations, and will write $pair \cdot a \cdot b$ when there is no danger of confusion.

In untyped models, pairing is automatic

$$pair = \lambda^* xyz.zxy, \quad fst = \lambda^* p.p(\lambda^* xy.x), \quad snd = \lambda^* p.p(\lambda^* xy.y)$$

1.3.3 Booleans

Definition 1.26. A model **A** has **booleans** if for some type $\mathtt{B}$ there exist elements

$$\begin{aligned} &\mathit{tt}, \mathit{ff} \in \mathbf{A}^\sharp(\mathtt{B}) \\ &\mathit{if}_\sigma \in \mathbf{A}(\mathtt{B}, \sigma, \sigma \to \sigma) \text{ for each } \sigma \end{aligned}$$

s.t. for all $x, y \in \mathbf{A}^\circ(\sigma)$ we have

$$\mathit{if}_\sigma \cdot \mathit{tt} \cdot x \cdot y = x, \quad \mathit{if}_\sigma \cdot \mathit{ff} \cdot x \cdot y = y$$

Note that tt, ff need not be the sole element of $\mathbf{A}^\sharp(\mathbf{B})$

Alternatively, we may define a notion of having booleans in the setting of computability model **C** with weak products: replace if_σ with $\mathit{if}'_\sigma \in \mathbf{C}[\mathbb{B} \times \sigma \times \sigma, \sigma]$. In a TPCA with products and pairing the two definitions coincide

In untyped models, the existence of booleans is automatic: $\mathit{tt} = \lambda^* xy.x$, $\mathit{ff} = \lambda^* xy.y$ and $\mathit{if} = \lambda^* zxy.zxy$

Obviously, the value of an expression $\mathit{if}_\sigma \cdot b \cdot e \cdot e'$ cannot be defined unless the values of both e and e' are defined. However, there is a useful trick that allows us to build conditional expressions whose definedness requires only that the chosen branch of the conditional is defined. This trick is specific to the higher-order setting, and is known as **strong definition by cases**:

Proposition 1.27. *Suppose **A** has booleans as above. Given applicative expressions $e, e' : \sigma$ there is an applicative expression $(e \mid e') : \mathtt{B} \to \sigma$ s.t. for any valuation v covering $V(e)$ and $V(e')$ we have*

$$[\![(e \mid e')]\!]_v \downarrow, \quad [\![(e \mid e') \cdot \mathit{tt}]\!]_v \succeq [\![e]\!]_v, \quad [\![(e \mid e') \cdot \mathit{ff}]\!]_v \succeq [\![e']\!]_v$$

Proof. Let ρ be any type s.t. $\mathbf{A}^\circ(\rho)$ is inhabited by some element a, and define

$$(e \mid e') = \lambda^* z^{\mathtt{B}} \cdot (\mathit{if}_\sigma z (\lambda^* r^\rho.e)(\lambda^* r^\rho.e') c_a)$$

where z, r are fresh variables

$[\![(e \mid e')]\!]_v \downarrow$ since by lax combinatory completeness

$[\![(e \mid e') \cdot \mathit{tt}]\!]_v \succeq [\![e]\!]_v$ by 1.21 □

The expressions $\lambda^* r.e$, $\lambda^* r.e'$ in the above proof are known as **suspensions** or **thunks**: the idea is that $[\![\lambda^* r.e]\!]_v$ is guaranteed to be defined, but the actual evaluation of e_v (which may be undefined) is 'suspended' until the argument c_a is supplied.

1.3.4 Numerals

Definition 1.28. A model **A** has **numerals** if for some type $\mathtt{N}$ there exist

$$\hat{0}, \hat{1}, \hat{2}, ... \in \mathbf{A}^\sharp(\mathtt{N})$$
$$suc \in \mathbf{A}^\sharp(\mathtt{N} \to \mathtt{N})$$

and for any $x \in \mathbf{A}^\sharp(\sigma)$ and $f \in \mathbf{A}^\sharp(\mathtt{N} \to \sigma \to \sigma)$ an element

$$Rec_\sigma(x, f) \in \mathbf{A}^\sharp(\mathtt{N} \to \sigma)$$

s.t. for all $x \in \mathbf{A}^\sharp(\sigma)$, $f \in \mathbf{A}^\sharp(\mathtt{N} \to \sigma \to \sigma)$ and $n \in \mathbb{N}$ we have

$$suc \cdot \hat{n} = \widehat{n+1}$$
$$Rec_\sigma(x, f) \cdot \hat{0} = x$$
$$Rec_\sigma(x, f) \cdot \widehat{n+1} \succeq f \cdot \hat{n} \cdot (Rec_\sigma(x, f) \cdot \hat{n})$$

The above definition has the advantage that it naturally adapts to the setting of a computability model C with products: just replace the types of f and $Rec_\sigma(x, f)$ above with $\mathbf{C}[\mathtt{N} \times \sigma, \sigma]$ and $\mathbf{C}[\mathtt{N}, \sigma]$ respectively.

Proposition 1.29. *A model **A** has numerals iff it has elements $\hat{n}$, suc as above and*

$$rec_\sigma \in \boldsymbol{A}^\sharp(\sigma \to (\mathtt{N} \to \sigma \to \sigma) \to \mathtt{N} \to \sigma) \quad \textit{for each } \sigma$$

s.t. for all $x \in \boldsymbol{A}^\circ(\sigma)$, $f \in \boldsymbol{A}^\circ(\mathtt{N} \to \sigma \to \sigma)$ and $n \in \mathbb{N}$ we have

$$suc \cdot \hat{n} = \widehat{n+1}$$
$$rec_\sigma \cdot x \cdot f \cdot \hat{0} = x$$
$$rec_\sigma \cdot x \cdot f \cdot \widehat{n+1} \succeq f \cdot \hat{n} \cdot (rec_\sigma \cdot x \cdot f \cdot \hat{n})$$

Proof. $\Leftarrow$: Let $Rec_\sigma(x, f) = rec_\sigma \cdot x \cdot f$
$\Rightarrow$: define

$$rec_\sigma = Rec_{\sigma \to (\mathtt{N} \to \sigma \to \sigma) \to \sigma}(\lambda^* xf.x, \lambda^* nr.\lambda^* xf.fn(rxf))$$

?

□

Exercise 1.3.1. Show that **A** has numerals, then **A** has booleans

Proposition 1.30. *Every untyped model has numerals*

Proof. Using the encodings for pairings and booleans given above, we may define the **Curry numerals** $\hat{n}$ in any untyped models as follows:

$$\hat{0} = \langle tt, tt \rangle, \quad \widehat{n+1} = \langle ff, \hat{n} \rangle$$

and $suc = \lambda^* x.\langle ff, x \rangle$. We also have elements for the zero testing and predecessor operations: take $iszero = fst$ and $pre = \lambda^* x.if(iszero\ x)\hat{0}(snd\ x)$ □

In any model with numerals, a rich class of functions $\mathbb{N}^r \to \mathbb{N}$ is representable. For example, the (first-order) primitive recursive functions on $\mathbb{N}$

Proposition 1.31. *For any primitive recursive $f : \mathbb{N}^r \to \mathbb{N}$ there is an applicative expression $e_f : \mathbb{N}^{(r)} \to \mathbb{N}$ (involving constants 0, suc, rec_N) s.t. in any model $(\boldsymbol{A}^\circ; \boldsymbol{A}^\sharp)$ with numerals we have $[\![e_f]\!]_v \in \boldsymbol{A}^\sharp$ (where v is the obvious valuation of the constants) and*

$$\forall n_0, \ldots, n_{r-1}, m. f(n_0, \ldots, n_{r-1}) = m \Rightarrow [\![e_f]\!]_v \cdot \hat{n}_0 \cdot \hat{n}_{r-1} = \hat{m}$$

Proof. □

1.3.5 Recursion and Minimization

Definition 1.32.

1. A total model **A has general recursion**, or **has fixed points**, if for every element $f \in \mathbf{A}^\sharp(\rho \to \rho)$ there is an element $Fix_\rho(f) \in \mathbf{A}^\sharp(\rho)$ s.t. $Fix_\rho(f) = f \cdot Fix_\rho(f)$
2. An arbitrary model **A has guarded recursion**, or **guarded fixed points**, if for every element $f \in \mathbf{A}^\sharp(\rho \to \rho)$ where $\rho = \sigma \to \tau$ there is an element $GFix_\rho(f) \in \mathbf{A}^\sharp(\rho)$ s.t. $GFix_\rho(f) \cdot x \succeq f \cdot GFix_\rho(f) \cdot x$ for all $x \in \mathbf{A}^\circ(\sigma)$

Proposition 1.33.

1. *A total model $\boldsymbol{A}$ has general recursion iff for every type ρ there is an element $Y_\rho \in \boldsymbol{A}^\sharp((\rho \to \rho) \to \rho)$ s.t. for all $f \in \boldsymbol{A}^\circ(\rho \to \rho)$ we have*
$$Y_\rho \cdot f = f \cdot (Y_\rho \cdot f)$$
2. *$\boldsymbol{A}$ has guarded recursion iff for every type $\rho = \sigma \to \tau$ there is an element $Z_\rho \in \boldsymbol{A}^\sharp((\rho \to \rho) \to \rho)$ s.t. for all $f \in \boldsymbol{A}^\circ(\rho \to \rho)$ and $x \in \boldsymbol{A}^\circ(\sigma)$ we have*
$$Z_\rho \cdot f \downarrow, \quad Z_\rho \cdot f \cdot x \succeq f \cdot (Z_\rho \cdot f) \cdot x$$

Proof. Define

$$Y_\rho = Fix_{(\rho\to\rho)\to\rho}(\lambda^* y.\lambda^* f.f(yf)), \quad Z_\rho = GFix_{(\rho\to\rho)\to\rho}(\lambda^* z.\lambda^* fx.f(zf)x)$$

□

Not all models of interest possess such recursion operators. Clearly, if **A** is a **total** model with $\mathbf{A}(\mathtt{N}) = \mathbb{N}$ a type of numerals as above, then **A** cannot have general or even guarded recursion: if $\rho = \mathtt{N} \to \mathtt{N}$ and $f = \lambda^* gx.suc(gx)$ then we would have $Z \cdot f \cdot \hat{n} = suc \cdot Z \cdot f \cdot \hat{n}$, which is impossible. However, many models with $\mathbf{A}(\mathtt{N}) = \mathbb{N}_\perp$ will have general recursion

Any **untyped** total model has general recursion, since we may take

$$W = \lambda^* wf.f(wwf), \quad Y = WW$$

(This element Y is known as the **Turing fixed point combinator**). Likewise, every untyped model, total or not, has guarded recursion, since we may take

$$V = \lambda^* vfx.f(vvf)x, \quad Z = VV$$

Note in passing that Kleene's **second recursion theorem** from classical computability theory is tantamount to the existence of a guarded recursion operator in K_1

We can now prove 1.30. In any untyped model, let Z be a guarded recursion operator, define

$$R = \lambda^* rxfm.if(iszero\ m)(kx)(\lambda^* y.f(pre\ m))(rxf(pre\ m)\hat{0})$$

and take $rec = \lambda^* xfm.(ZR)xfmi$.

Definition 1.34. A model **A** with numerals **has minimization** if it contains an element $min \in \mathbf{A}^\sharp((\mathtt{N} \to \mathtt{N}) \to \mathtt{N})$ s.t. whenever $\hat{g} \in \mathbf{A}^\circ(\mathtt{N} \to \mathtt{N})$ represents some total $g : \mathbb{N} \to \mathbb{N}$ and m is the least number s.t. $g(m) = 0$, we have $min \cdot \hat{g} = \hat{m}$

Proposition 1.35. *There is an applicative expression Min involving constants* $\hat{0}$*, suc, iszero, if and Z s.t. in any model with numerals and guarded recursion,* $[\![Min]\!]_v$ *is a minimization operator*

Proof. Take $Min = Z(\lambda^* M.\lambda^* g.if(iszero(g\ \hat{0}))\hat{0}(M(\lambda^* n.g(suc\ n))))$ □

Proposition 1.36. *For any partial computable* $f : \mathbb{N}^r \rightharpoonup \mathbb{N}$ *there is an applicative expression* $e_f : \mathtt{N}^{(r)} \to \mathtt{N}$ *(involving constants 0, suc,* $rec_{\mathtt{N}}$*, min) s.t. in any model* **A** *with numerals and minimization we have* $[\![e_f]\!]_v \in \mathbf{A}^\sharp$ *(with the obvious valuation* v*) and*

$$\forall n_0, \ldots, n_{r-1}, m.f(n_0, \ldots, n_{r-1}) = m \Rightarrow [\![e_f]\!]_v \cdot \hat{n}_0 \cdot \cdots \cdot \hat{n}_{r-1} = \hat{m}$$

Proof. Since our definition of minimization refers only to total functions $g : \mathbb{N} \to \mathbb{N}$, we appeal to the *Kleene normal form* theorem: there are primitive recursive functions $T : \mathbb{N}^{r+2} \to \mathbb{N}$ and $U : \mathbb{N} \to \mathbb{N}$ such that any partial computable f has an 'index' $e \in \mathbb{N}$ such that $f(\bar{n}) \simeq U(\mu y.T(e, \bar{n}, y) = 0)$ for all $\bar{n}$. Using this, the result follows easily from Propositions 1.31 and 1.35. □

1.3.6 The Category of Assemblies

Definition 1.37. Let **C** be a lax computability model over T. The **category of assemblies over C**, written $\mathcal{A}\mathrm{sm}(\mathbf{C})$ is defined as follows:

- Objects X are triples $(|X|, \rho_X, \Vdash_X)$ where $|X|$ is a set, $\rho_X \in T$ names some type, and $\Vdash_X \subseteq \mathbf{C}(\rho_X) \times |X|$ is a relation s.t. $\forall x \in |X|.\exists a \in \mathbf{C}(\rho_X).a \Vdash_X x$ (The formula $a \Vdash_X x$ may be read as 'a **realizes** x')
- A morphism $f : X \to Y$ is a function $f : |X| \to |Y|$ that is **tracked** by some $\bar{f} \in \mathbf{C}[\rho_X, \rho_Y]$, in the sense that for any $x \in |X|$ and $a \in \mathbf{C}(\rho_X)$ we have
$$a \Vdash_X x \Rightarrow \bar{f}(a) \Vdash_Y f(x)$$

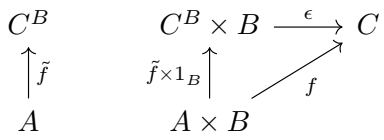
An assembly X is called **modest** if $a \Vdash_X x \wedge a \Vdash_X x'$ implies $x = x'$. We write $\mathcal{M}\mathrm{od}(\mathbf{C})$ for the full subcategory of $\mathcal{A}\mathrm{sm}(\mathbf{C})$ consisting of modest assemblies

Intuitively, we regard an assembly X as an "abstract datatype" for which we have a concrete implementation on the "machine" **C**. The underlying set $|X|$ is the set of values of the abstract type, and for each $x \in |X|$, the elements $a \Vdash_X x$ are the possible machine representations of this abstract value. (Note that an abstract value x may have many possible machine representations a.) The morphisms $f : X \to Y$ may then be regarded as the "computable mappings" between such datatypes

In the case that **C** is a lax TPCA **A**, we may also denote the above categories $\mathcal{A}\mathrm{sm}(\mathbf{A})$, $\mathcal{M}\mathrm{od}(\mathbf{A})$, or by $\mathcal{A}\mathrm{sm}(\mathbf{A}^\circ; \mathbf{A}^\sharp)$, $\mathcal{M}\mathrm{od}(\mathbf{A}^\circ; \mathbf{A}^\sharp)$. Note that realizers for elements $x \in |X|$ may be arbitrary elements of $\mathbf{A}^\circ(\rho_X)$, whereas a morphism $f : X \to Y$ must be tracked by an element of $\mathbf{A}^\sharp(\rho_X \to \rho_Y)$

Viewed in this way, all the datatypes we shall typically wish to consider in fact live in the subcategory $\mathcal{M}\mathrm{od}(\mathbf{C})$: an abstract data value is uniquely determined by any of its machine representations. Note also that if Y is modest, a morphism $f : X \to Y$ is completely determined by any $\bar{f}$ that tracks it.

Definition 1.38. Let the category **C** have binary products. An **exponential** of objects B and C consists of an object C^B and an arrow $\epsilon : C^B \times B \to C$ s.t. for any object A and arrow $f : A \times B \to C$ there is a unique arrow $\tilde{f} : A \to C^B$ s.t. $\epsilon \circ (\tilde{f} \times 1_B) = f$



Theorem 1.39. *Let **C** be a lax computability model*

1. *If **C** has a weak terminal, then $\mathcal{A}\mathrm{sm}(\boldsymbol{C})$ has a terminal object 1*
2. *If **C** has weak products, then $\mathcal{A}\mathrm{sm}(\boldsymbol{C})$ has binary cartesian products*
3. *If **C** weakly cartesian closed, then $\mathcal{A}\mathrm{sm}(\boldsymbol{C})$ is cartesian closed*
4. *If **C** has a weak terminal and booleans, $\mathcal{A}\mathrm{sm}(\boldsymbol{C})$ has the coproduct $1 + 1$*
5. *If **C** has a weak terminal and numerals, $\mathcal{A}\mathrm{sm}(\boldsymbol{C})$ has a natural number object*

Proof. 1. If (I, i) is a weak terminal, define $1 = (\{i\}, I, \Vdash_1 = \{(i, i)\})$. Then for any $X \in \mathcal{A}\mathrm{sm}(\mathbf{C})$, $f = \Lambda x.i$ is the unique morphism where $\overline{f} = \Lambda x.i$.

2. If X and Y are assemblies and ρ is a weak product of ρ_X and ρ_Y, define the assembly $X \times Y$ by

$$|X \times Y| = |X| \times |Y|, \quad \rho_{X \times Y} = \rho, \quad a \Vdash_{X \times Y} (x, y) \text{ iff } \pi_X(a) \Vdash_X x \wedge \pi_Y(a) \Vdash_Y y$$

3. If X and Y are assemblies, let us say an element $t \in \mathbf{C}(\rho_X \to \rho_Y)$ **tracks** a function $f : |X| \to |Y|$ if

$$\forall x \in |X|, a \in \mathbf{C}(\rho_X).\ a \Vdash_X x \Rightarrow t \cdot_{XY} a \Vdash_Y f(x)$$

Now define the assembly Y^X as follows:

$$\begin{aligned} |Y^X| &= \{f : |X| \to |Y| \mid f \text{ is tracked by some } t \in \mathbf{C}(\rho_X \to \rho_Y)\} \\ \rho_{Y^X} &= \rho_X \to \rho_Y \\ t \Vdash_{Y^X} f &\Leftrightarrow t \text{ tracks } f \end{aligned}$$

□

Theorem **??** also holds with $\mathcal{M}\mathrm{od}(\mathbf{C})$, and the inclusion $\mathcal{M}\mathrm{od}(\mathbf{C}) \hookrightarrow \mathcal{A}\mathrm{sm}(\mathbf{C})$ preserves all the relevant structure