2. Free Algebras, Recursion

We need to set up the number systems, that is we have to construct natural numbers, integers, rationals and reals. For efficiency reasons the natural numbers will be given in binary form. We start with One and use the constructors SZero and SOne; this will give a close correspondence to the usual way to write numbers in binary. Integers are generated from IntZero by the constructors IntPos and IntNeg. A rational is generated from an integer (the numerator) and a positive (the denominator) by RatConstr.

Generally, a free algebra is given by *constructors*, for instance zero and successor for the natural numbers. We want to treat other data types as well, like lists and binary trees.

2.1. Examples of Finitary and Infinitary Algebras.

(1) The easiest example is the algebra $\mathbf{U}$, which has just one nullary constructor called Dummy. It consists of exactly one element.

(2) Almost as easy is the next example, the algebra $\mathbf{B}$, which has just two nullary constructors called tt and ff (sometimes we use `True` and `False` instead). It consists of exactly two elements.

(3) The next example is the simplest algebra with infinitely many elements, the algebra $\mathbf{N}$ of natural numbers. Its constructors are a nullary constructor 0 (also written `Zero`) and a unary constructor S (also written `Succ`).

(4) We shall also allow parametrized algebras, depending on type parameters. The simplest example is the algebra $\mathbf{L}(\rho)$ (sometimes we use (`list` ρ) instead) of lists of objects of a given type ρ. Its constructors are a nullary constructor Nil and a binary constructor Cons; the latter takes an object x of type ρ and a list l and constructs out of these the new list obtained by putting x to the front of l.

(5) Another important parametrized algebra is the disjoint union of the algebras given by types ρ_1 and ρ_2. It is called (ρ_1 `yplus` ρ_2) ("y" from t*y*pe), and is given by two unary constructors Inl and Inr of types $\rho_1 \to \rho_1 + \rho_2$ and $\rho_2 \to \rho_1 + \rho_2$, respectively.

(6) We also admit simultaneously generated free algebras. An example is provided by the obvious way to simultaneously generate trees and finite lists (of arbitrary length) of trees. The algebra tree has one nullary constructor Leaf (taking a natural number as parameter) and one unary constructor Branch, building a tree from a tree list. The algebra tlist has one nullary constructor Empty and one binary constructor Tcons; the latter takes an object t of type tree and a tree list l and constructs out of these the new tree list obtained by putting t to the front of l.

All these algebras are *finitary*, in the sense that every constructor takes only finitely many arguments. Then for any two elements of the algebra we can decide whether they are equal.

We also allow *infinitary* free algebras, whose constructors may take infinitely many arguments. Notice that then equality is *not* decidable any more, and hence needs to be axiomatized.

For an example of an infinitary algebra, consider the countable ordinals. They can be seen as generated from a nullary constructor 0, a unary constructor for the successor and a constructor building the supremum out of a countably infinite list of ordinals (given by a function from the natural numbers to ordinals).

2.2. **Examples for Recursive Definitions.** Clearly the generation process of the elements of a free algebra allows *recursive* definitions of functions on such algebras. For the examples above, they have (when no parameters are present) the following form.

(1) For the algebra $\mathbf{U}$ we may just define
$$f(\mathsf{Dummy}) = a$$

(2) Similarly for the algebra $\mathbf{B}$ we can explicitely define
$$f(\mathsf{tt}) = a,$$
$$f(\mathsf{ff}) = b.$$

(3) For the algebra $\mathbf{N}$ of natural numbers we obtain the familiar recursion scheme
$$f(0) = a,$$
$$f(\mathsf{S}(n)) = h(n, f(n)).$$

(4) For the parametrized algebra $\mathbf{L}(\rho)$ we have a very similar recursion scheme:
$$f(\mathsf{Nil}) = a,$$
$$f(\mathsf{Cons}(x, l)) = h(x, l, f(l)).$$

(5) For the disjoint union of given types ρ_1 and ρ_2, the algebra does not require recursive calls, and hence the recursion scheme is simply
$$f(\mathsf{Inl}(x)) = g(x),$$
$$f(\mathsf{Inr}(y)) = h(y).$$

(6) Somewhat more interesting are simultaneously generated free algebras. Here we may define
$$f(\mathsf{Leaf}(n)) = h_1(n),$$
$$f(\mathsf{Branch}(l)) = h_2(l, g(l)),$$
$$g(\mathsf{Empty}) = b,$$
$$g(\mathsf{Tcons}(x, l)) = h_3(x, l, f(x), g(l)).$$

Further examples can be given for operations on the positive numbers.

2.3. **Inductively Generated Types.** Our type system is defined by three type forming operations: arrow types $\rho \to \sigma$, pair types $\rho \times \sigma$ and the formation of inductively generated types $\mu \vec{\alpha}\, \vec{\kappa}$, where $\vec{\alpha} = (\alpha_j)_{j=1,\dots,N}$ is a list of distinct "type variables", and $\vec{\kappa} = (\kappa_i)_{i=1,\dots,k}$ is a list of "constructor types", whose argument types contain $\alpha_1, \dots, \alpha_N$ in strictly positive positions only.

For instance, $\mu\alpha(\alpha, \alpha \to \alpha)$ is the type of natural numbers; here the list $(\alpha, \alpha \to \alpha)$ stands for two generation principles: α for "there is a natural number" (the 0), and $\alpha \to \alpha$ for "for every natural number there is another one" (its successor).

Let an infinite supply of *type variables* α, β be given.

Definition. Let $\vec{\alpha} = (\alpha_j)_{j=1,\dots,N}$ be a list of distinct type variables. *Types* $\rho, \sigma, \tau, \mu \in \mathsf{T}$ and *constructor types* $\kappa \in \mathsf{KT}(\vec{\alpha})$ are defined inductively as follows.

$$\frac{\vec{\rho}, \vec{\sigma}_1, \dots, \vec{\sigma}_n \in \mathsf{T}}{\vec{\rho} \to (\vec{\sigma}_1 \to \alpha_{j_1}) \to \dots \to (\vec{\sigma}_n \to \alpha_{j_n}) \to \alpha_j \in \mathsf{KT}(\vec{\alpha})} \quad (n \geq 0)$$

$$\frac{\kappa_1, \dots, \kappa_n \in \mathsf{KT}(\vec{\alpha})}{(\mu\vec{\alpha}\,(\kappa_1, \dots, \kappa_n))_j \in \mathsf{T}} \quad (n \geq 1) \qquad \frac{\rho, \sigma \in \mathsf{T}}{\rho \to \sigma \in \mathsf{T}} \qquad \frac{\rho, \sigma \in \mathsf{T}}{\rho \times \sigma \in \mathsf{T}}$$

Here $\vec{\rho}$ is short for a list $\rho_1, \dots, \rho_m$ $(m \geq 0)$ of types and $\vec{\rho} \to \sigma$ means $\rho_1 \to \dots \to \rho_m \to \sigma$, associated to the right. We shall use μ for types of the form $(\mu\vec{\alpha}\,(\kappa_1, \dots, \kappa_k))_j$ only, and for types $\vec{\tau}$ and a constructor type $\kappa = \vec{\rho} \to (\vec{\sigma}_1 \to \alpha_{j_1}) \to \dots \to (\vec{\sigma}_n \to \alpha_{j_n}) \to \alpha_j \in \mathsf{KT}(\vec{\alpha})$ let

$$\kappa[\vec{\tau}] := \vec{\rho} \to (\vec{\sigma}_1 \to \tau_{j_1}) \to \dots \to (\vec{\sigma}_n \to \tau_{j_n}) \to \tau_j.$$

Examples.

$$\begin{aligned}
&\mathbf{U} && := \mu\alpha\,\alpha,\\
&\mathbf{B} && := \mu\alpha\,(\alpha, \alpha),\\
&\mathbf{N} && := \mu\alpha\,(\alpha, \alpha \to \alpha),\\
&\mathbf{L}(\rho) && := \mu\alpha\,(\alpha, \rho \to \alpha \to \alpha),\\
&\rho_1 + \rho_2 && := \mu\alpha\,(\rho_1 \to \alpha, \rho_2 \to \alpha),\\
&(\mathsf{tree}, \mathsf{tlist}) && := \mu(\alpha, \beta)\,(\mathbf{N} \to \alpha, \beta \to \alpha, \beta, \alpha \to \beta \to \beta),\\
&\mathbf{Bin} && := \mu\alpha\,(\alpha, \alpha \to \alpha \to \alpha),\\
&\mathcal{O} && := \mu\alpha\,(\alpha, \alpha \to \alpha, (\mathbf{N} \to \alpha) \to \alpha),\\
&\mathcal{T}_0 && := \mathbf{N},\\
&\mathcal{T}_{n+1} && := \mu\alpha\,(\alpha, (\mathcal{T}_n \to \alpha) \to \alpha).
\end{aligned}$$

Notice that there are many equivalent ways to define these types. For instance, we could take $\mathbf{U} + \mathbf{U}$ to be the type of booleans, and $\mathbf{L}(\mathbf{U})$ to be the type of natural numbers.

Notice also that we have added the pair type $\rho \times \sigma$ for simplicity only. Products could have been defined in two forms, as tensor products and as cartesian products, by

$$\begin{aligned}
\rho_1 \otimes \rho_2 &:= \mu\alpha.\rho_1 \to \rho_2 \to \alpha,\\
\rho_1 \times_\sigma \rho_2 &:= \mu\alpha.(\sigma \to \rho_1) \to (\sigma \to \rho_2) \to \sigma \to \alpha,
\end{aligned}$$

If we would allow ourselves to quantify over types, the cartesian product could be defined as

$$\rho_1 \times \rho_2 := \mu\alpha\forall\sigma.(\sigma \to \rho_1) \to (\sigma \to \rho_2) \to \sigma \to \alpha.$$

2.4. **Constructors, Recursion Operators.** The inductive structure of the types $\vec{\mu} = \mu\vec{\alpha}\,\vec{\kappa}$ corresponds to two sorts of constants. With the *constructors* $\mathsf{constr}_i^{\vec{\mu}} \colon \kappa_i[\vec{\mu}]$ we can construct elements of a type μ_j, and with the *recursion operators* $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$ we can construct mappings from μ_j to τ_j by recursion on the structure of $\vec{\mu}$. In order to define the type of the recursion operators w.r.t. $\vec{\mu} = \mu\vec{\alpha}\,\vec{\kappa}$ and result types $\vec{\tau}$, we first define for

$$\kappa_i = \vec{\rho} \to (\vec{\sigma}_1 \to \alpha_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \alpha_{j_n}) \to \alpha_j \in \mathsf{KT}(\vec{\alpha})$$

the *step type*

$$\begin{aligned}\delta_i^{\vec{\mu},\vec{\tau}} := \vec{\rho} \to (\vec{\sigma}_1 \to \mu_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \mu_{j_n}) \to \\ (\vec{\sigma}_1 \to \tau_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \tau_{j_n}) \to \tau_j.\end{aligned}$$

Here $\vec{\rho}, (\vec{\sigma}_1 \to \mu_{j_1}), \dots, (\vec{\sigma}_n \to \mu_{j_n})$ correspond to the *components* of the object of type μ_j under consideration, and $(\vec{\sigma}_1 \to \tau_{j_1}), \dots, (\vec{\sigma}_n \to \tau_{j_n})$ to the previously defined values. The recursion operator $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$ has type

$$\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}} \colon \delta_1^{\vec{\mu},\vec{\tau}} \to \cdots \to \delta_k^{\vec{\mu},\vec{\tau}} \to \mu_j \to \tau_j$$

(recall that k is the total number of constructors for all types $\mu_1, \dots, \mu_N$).

We will often write $\mathcal{R}_j^{\vec{\mu},\vec{\tau}}$ for $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$, and omit the upper indices $\vec{\mu}, \vec{\tau}$ when they are clear from the context. In case of a non-simultaneous free algebra, i.e. of type $\mu\alpha\,\kappa$, for $\mathcal{R}_\mu^{\mu,\tau}$ we normally write $\mathcal{R}_\mu^\tau$.

2.5. **Examples for Recursive Definitions and Recursion Operators.** We begin with some examples for constructors and recursion operators.

$$\begin{aligned}
&\mathsf{tt}^{\mathbf{B}} := \mathsf{constr}_1^{\mathbf{B}}, \quad \mathsf{ff}^{\mathbf{B}} := \mathsf{constr}_2^{\mathbf{B}},\\
&\mathcal{R}_{\mathbf{B}}^\tau \colon \tau \to \tau \to \mathbf{B} \to \tau,\\[1ex]
&0^{\mathbf{N}} := \mathsf{constr}_1^{\mathbf{N}}, \quad \mathsf{S}^{\mathbf{N}\to\mathbf{N}} := \mathsf{constr}_2^{\mathbf{N}},\\
&\mathcal{R}_{\mathbf{N}}^\tau \colon \tau \to (\mathbf{N} \to \tau \to \tau) \to \mathbf{N} \to \tau,\\[1ex]
&\mathsf{Nil}^{\mathbf{L}(\alpha)} := \mathsf{constr}_1^{\mathbf{L}(\alpha)}, \quad \mathsf{Cons}^{\alpha\to\mathbf{L}(\alpha)\to\mathbf{L}(\alpha)} := \mathsf{constr}_2^{\mathbf{L}(\alpha)},\\
&\mathcal{R}_{\mathbf{L}(\alpha)}^\tau \colon \tau \to (\alpha \to \mathbf{L}(\alpha) \to \tau \to \tau) \to \mathbf{L}(\alpha) \to \tau,\\[1ex]
&\big(\mathsf{Inl}_{\rho_1\rho_2}\big)^{\rho_1\to\rho_1+\rho_2} := \mathsf{constr}_1^{\rho_1+\rho_2},\\
&\big(\mathsf{Inr}_{\rho_1\rho_2}\big)^{\rho_2\to\rho_1+\rho_2} := \mathsf{constr}_2^{\rho_1+\rho_2},\\
&\mathcal{R}_{\rho_1+\rho_2}^\tau \colon (\rho_1 \to \tau) \to (\rho_2 \to \tau) \to \rho_1 + \rho_2 \to \tau.
\end{aligned}$$

Notice that for the defined products the constructors and recursion operators are

$$\begin{aligned}
&\big(\otimes_{\rho_1\rho_2}^+\big)^{\rho_1\to\rho_2\to\rho_1\otimes\rho_2} := \mathsf{constr}_1^{\rho_1\otimes\rho_2},\\
&\mathcal{R}_{\rho_1\otimes\rho_2}^\tau \colon (\rho_1 \to \rho_2 \to \tau) \to \rho_1\otimes\rho_2 \to \tau,\\[1ex]
&\big(\times_{\rho_1\rho_2\sigma}^+\big)^{(\sigma\to\rho_1)\to(\sigma\to\rho_2)\to\sigma\to\rho_1\times_\sigma\rho_2} := \mathsf{constr}_1^{\rho_1\times_\sigma\rho_2},\\
&\mathcal{R}_{\rho_1\times_\sigma\rho_2}^\tau \colon ((\sigma \to \rho_1) \to (\sigma \to \rho_2) \to \sigma \to \tau) \to \rho_1 \times_\sigma \rho_2 \to \tau,
\end{aligned}$$

Here are some examples for recusive definitions and the corresponding recursion operators. The *append*-function :+: for lists is defined recursively by

$$\begin{aligned}
\mathsf{Nil} :+: l_2 &:= l_2,\\
(\mathsf{Cons}\,x\,l_1) :+: l_2 &:= \mathsf{Cons}\,x(l_1 :+: l_2).
\end{aligned}$$

It can be defined as the term

$$M_{:+:} := \mathcal{R}_{\mathbf{L}(\alpha)}^{\mathbf{L}(\alpha)\to\mathbf{L}(\alpha)}(\lambda l_2 l_2)(\lambda x\lambda l_1\lambda p\lambda l_2.\mathsf{Cons}\,x(pl_2)).$$

Using the append function :+: we can define *list reversal* Rev by

$$\mathsf{Rev}\,\mathsf{Nil} := \mathsf{Nil},$$
$$\mathsf{Rev}(\mathsf{Cons}\,x\,l) = (\mathsf{Rev}\,l) :+: (\mathsf{Cons}\,x\mathsf{Nil}).$$

It can be defined as the term

$$\mathcal{R}_{\mathbf{L}(\alpha)}^{\mathbf{L}(\alpha)}\mathsf{Nil}(\lambda x\lambda l\lambda p.M_{:+:}\,p\,(\mathsf{Cons}\,x\mathsf{Nil})).$$

Assume we want to define by simultaneous recursion two functions on $\mathbf{N}$, say $\mathsf{Even},\mathsf{Odd}\colon \mathbf{N}\to\mathbf{B}$. We want

$$\mathsf{Even}(0) := \mathsf{tt} \qquad \mathsf{Odd}(0) := \mathsf{ff}$$
$$\mathsf{Even}(\mathsf{S}\,n) := \mathsf{Odd}(n) \qquad \mathsf{Odd}(\mathsf{S}\,n) := \mathsf{Even}(n)$$

This can be achieved by using pair types: we recursively define the single function $\mathsf{EvenOdd}\colon \mathbf{N}\to\mathbf{B}\times\mathbf{B}$. The step types are

$$\delta_1 = \mathbf{B}\times\mathbf{B},$$
$$\delta_2 = \mathbf{N}\to\mathbf{B}\times\mathbf{B}\to\mathbf{B}\times\mathbf{B},$$

and we can define $\mathsf{EvenOdd} := \mathcal{R}_{\mathbf{N}}^{\mathbf{B}\times\mathbf{B}}\langle\mathsf{ff},\mathsf{tt}\rangle(\lambda n\lambda p.\langle p1, p0\rangle)$.

Our final example concerns the simultaneously defined free algebras tree and tlist, whose constructors $\mathsf{constr}_i^{(\mathsf{tree},\mathsf{tlist})}$ for $i\in\{1,\dots,4\}$ are

$$\mathsf{Leaf}^{\mathbf{N}\to\mathsf{tree}}, \mathsf{Branch}^{\mathsf{tlist}\to\mathsf{tree}}, \mathsf{Empty}^{\mathsf{tlist}}, \mathsf{Tcons}^{\mathsf{tree}\to\mathsf{tlist}\to\mathsf{tlist}}.$$

Observe that the elements of the algebra tree are just the finitely branching trees, which carry natural numbers on their leaves.

Let us compute the types of the recursion operators w.r.t. the result types τ_1,τ_2, i.e. of $\mathcal{R}_{\mathsf{tree}}^{(\mathsf{tree},\mathsf{tlist}),(\tau_1,\tau_2)}$ and $\mathcal{R}_{\mathsf{tlist}}^{(\mathsf{tree},\mathsf{tlist}),(\tau_1,\tau_2)}$, or shortly $\mathcal{R}_{\mathsf{tree}}$ and $\mathcal{R}_{\mathsf{tlist}}$. The step types are

$$\delta_1 := \mathbf{N}\to\tau_1,$$
$$\delta_2 := \mathsf{tlist}\to\tau_2\to\tau_1,$$
$$\delta_3 := \tau_2,$$
$$\delta_4 := \mathsf{tree}\to\mathsf{tlist}\to\tau_1\to\tau_2\to\tau_2.$$

Hence the types of the recursion operators are

$$\mathcal{R}_{\mathsf{tree}}\colon \delta_1\to\delta_2\to\delta_3\to\delta_4\to\mathsf{tree}\to\tau_1,$$
$$\mathcal{R}_{\mathsf{tlist}}\colon \delta_1\to\delta_2\to\delta_3\to\delta_4\to\mathsf{tlist}\to\tau_2.$$

To see a concrete example, let us recursively define addition $+\colon \mathsf{tree}\to\mathsf{tree}\to\mathsf{tree}$ and $\oplus\colon \mathsf{tlist}\to\mathsf{tree}\to\mathsf{tlist}$. The recursion equations to be satisfied are

$$+\,(\mathsf{Leaf}\,n) = \lambda a a,$$
$$+\,(\mathsf{Branch}\,bs) = \lambda a.\mathsf{Branch}(\oplus\,bs\,a),$$

$$\oplus\ \mathsf{Empty} = \lambda a\,\mathsf{Empty},$$
$$\oplus\,(\mathsf{Tcons}\,b\,bs) = \lambda a.\mathsf{Tcons}(+\,b\,a)(\oplus\,bs\,a).$$

We define $+$ and $\oplus$ by means of the recursion operators $\mathcal{R}_{\mathsf{tree}}$ and $\mathcal{R}_{\mathsf{tlist}}$ with result types

$$\tau_1 := \mathsf{tree} \to \mathsf{tree},$$
$$\tau_2 := \mathsf{tree} \to \mathsf{tlist}.$$

The step terms are

$$M_1 := \lambda n \lambda a a,$$
$$M_2 := \lambda bs \lambda g^{\tau_2} \lambda a.\mathsf{Branch}(g\, a),$$
$$M_3 := \lambda a\, \mathsf{Empty},$$
$$M_4 := \lambda b \lambda bs \lambda f^{\tau_1} \lambda g^{\tau_2} \lambda a.\mathsf{Tcons}(f\, a)(g\, a).$$

Then

$$+ := \mathcal{R}_{\mathsf{tree}} \vec{M} \colon \mathsf{tree} \to \mathsf{tree} \to \mathsf{tree},$$
$$\oplus := \mathcal{R}_{\mathsf{tlist}} \vec{M} \colon \mathsf{tlist} \to \mathsf{tree} \to \mathsf{tlist}.$$

2.6. **Special Cases of Recursion.** It may happen that in a recursion on simultaneously defined algebras one only needs to recur on some of those algebras. Then we can simplify the type of the recursion operator accordingly, by

- omitting all step types $\delta_i^{\vec{\mu},\vec{\tau}}$ with irrelevant value type τ_j, and
- simplifying the remaining step types by omitting from the types $(\vec{\sigma}_1 \to \tau_{j_1}), \dots, (\vec{\sigma}_n \to \tau_{j_n})$ of previously defined values all those with irrelevant τ_{j_ν}.

In the $\mathsf{tree}, \mathsf{tlist}$-example, if we only want to recur on tlist, then the step types are

$$\delta_3 := \tau_2,$$
$$\delta_4 := \mathsf{tree} \to \mathsf{tlist} \to \tau_2 \to \tau_2.$$

Hence the type of the simplified recursion operator is

$$\mathcal{R}_{\mathsf{tlist}} \colon \delta_3 \to \delta_4 \to \mathsf{tlist} \to \tau_2.$$

An example is the recursive definition of the length of a tlist. The recursion equations are

$$\mathsf{len}(\mathsf{Empty}) \quad = 0,$$
$$\mathsf{len}(\mathsf{Tcons}\, b\, bs) = \mathsf{len}(bs) + 1.$$

The step terms are

$$M_3 := \mathsf{Empty},$$
$$M_4 := \lambda b \lambda bs \lambda p.p + 1.$$

There is an important variant of recursion, where no recursive calls occur. This variant is called the *cases operator*; it distinguishes cases according to the outer constructor form. Here all step types have the form

$$\delta_i^{\vec{\mu},\vec{\tau}} := \vec{\rho} \to (\vec{\sigma}_1 \to \mu_{j_1}) \to \dots \to (\vec{\sigma}_n \to \mu_{j_n}) \to \tau_j.$$

The intended meaning of the cases operator is given by the conversion rule (cf. (5) below)

$$(\mathcal{C}_j \vec{M})^{\mu_j \to \tau_j} (\mathsf{constr}_i^{\vec{\mu}} \vec{N}) \mapsto M_i \vec{N}.$$

Notice that only those step terms are used whose value type is the present τ_j; this is due to the fact that there are no recursive calls. Therefore the type of the cases operator is

$$\mathcal{C}^{\vec{\mu}}_{\mu_j \to \tau_j} \colon \delta_{i_1} \to \cdots \to \delta_{i_q} \to \mu_j \to \tau_j,$$

where $\delta_{i_1}, \dots, \delta_{i_q}$ consists of all δ_i with value type τ_j. We write $\mathcal{C}_{\mu_j \to \tau_j}$ or even $\mathcal{C}_j$ for $\mathcal{C}^{\vec{\mu}}_{\mu_j \to \tau_j}$.

The simplest example (for the type $\mathbf{B}$) is *if-then-else*. Another example is the predecessor function on $\mathbf{N}$, i.e. $\mathsf{P}(0) := 0$, $\mathsf{P}(\mathsf{S}(n)) := n$. It can formally be defined by the term

$$\mathcal{C}_{\mathbf{N} \to \mathbf{N}} 0 (\lambda n n).$$

In the $\mathsf{tree}, \mathsf{tlist}$-example we have

$$\mathcal{C}_{\mathsf{tlist} \to \tau_2} \colon \tau_2 \to (\mathsf{tree} \to \mathsf{tlist} \to \tau_2) \to \mathsf{tlist} \to \tau_2.$$

Remark. When computing the value of a cases operator, we may not want to (eagerly) evaluate all arguments, but rather evaluate the test argument first and depending on the result (lazily) evaluate at most one of the other arguments. This phenomenon is well known in functional languages; e.g. in SCHEME the `if`-construct is called a *special form* (as opposed to an operator). Therefore we also provide an `if`-construct to build terms, which differs from the cases operator only in that it employs lazy evaluation. The predecessor function could then be written in the form $\lambda m[\mathtt{if}\ m\ 0\ \lambda n n]$.

2.7. **Terms, Conversion.** It will be useful to employ the following notation. Let $\vec{\mu} = \mu \vec{\alpha}\, \vec{\kappa}$ and

$$\kappa_i = \rho_1 \to \cdots \to \rho_m \to (\vec{\sigma}_1 \to \alpha_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \alpha_{j_n}) \to \alpha_j \in \mathsf{KT}(\vec{\alpha}),$$

and consider $\mathsf{constr}^{\vec{\mu}}_i \vec{N}$. Then we write $\vec{N}^P = N^P_1, \dots, N^P_m$ for the *parameter arguments* $N^{\rho_1}_1, \dots, N^{\rho_m}_m$ and $\vec{N}^R = N^R_1, \dots, N^R_n$ for the *recursive arguments* $N^{\vec{\sigma}_1 \to \mu_{j_1}}_{m+1}, \dots, N^{\vec{\sigma}_n \to \mu_{j_n}}_{m+n}$, and n^R for the number n of recursive arguments.

We define a *conversion relation* $\mapsto_\rho$ between terms of type ρ by

$$(\lambda x M) N \mapsto M[x := N] \tag{1}$$

$$\langle M_0, M_1 \rangle i \mapsto M i \quad (i = 0, 1) \tag{2}$$

$$\lambda x. M x \mapsto M \quad \text{if } x \notin \mathsf{FV}(M) \ (M \text{ not an abstraction}) \tag{3}$$

$$\langle M0, M1 \rangle \mapsto M \quad (M \text{ not a pair}) \tag{4}$$

$$(\mathcal{R}_j \vec{M})^{\mu_j \to \tau_j} (\mathsf{constr}^{\vec{\mu}}_i \vec{N}) \mapsto M_i \vec{N} \big((\mathcal{R}_{j_1} \vec{M}) \circ N^R_1 \big) \dots \big((\mathcal{R}_{j_n} \vec{M}) \circ N^R_n \big) \tag{5}$$

Here we have written $\mathcal{R}_j$ for $\mathcal{R}^{\vec{\mu}, \vec{\tau}}_{\mu_j}$.

The *one step reduction relation* $\to$ can now be defined as follows. $M \to N$ if N is obtained from M by replacing a subterm M' in M by N', where $M' \mapsto N'$. The reduction relations $\to^+$ and $\to^*$ are the transitive and the reflexive transitive closure of $\to$, respectively. For $\vec{M} = M_1, \dots, M_n$ we write $\vec{M} \to \vec{M}'$ if $M_i \to M'_i$ for some $i \in \{1, \dots, n\}$ and $M_j = M'_j$ for all $i \neq j \in \{1, \dots, n\}$. A term M is *normal* (or in *normal form*) if there is no term N such that $M \to N$.

Clearly normal closed terms are of the form $\mathsf{constr}^{\vec{\mu}}_i \vec{N}$.

Let us check the conversion rules for the defined $+$ and $\oplus$ of our example above. We have

$$+\,(\mathsf{Leaf}\,n) = \mathcal{R}_{\mathsf{tree}}\vec{M}\,(\mathsf{Leaf}\,n) \mapsto M_1 n \mapsto \lambda a a,$$
$$\oplus\ \mathsf{Empty} = \mathcal{R}_{\mathsf{tlist}}\vec{M}\ \mathsf{Empty} \mapsto M_3 = \lambda a\,\mathsf{Empty}$$

and

$$\begin{aligned} +(\mathsf{Branch}\,bs) &= \mathcal{R}_{\mathsf{tree}}\vec{M}\,(\mathsf{Branch}\,bs) \\ &\mapsto M_2\ bs\ \big((\mathcal{R}_{\mathsf{tlist}}\vec{M}) \circ bs\big) \\ &= M_2\ bs\ (\oplus\ bs) \\ &\to \lambda a.\mathsf{Branch}(\oplus\ bs\ a), \end{aligned}$$

$$\begin{aligned} \oplus(\mathsf{Tcons}\,b\,bs) &= \mathcal{R}_{\mathsf{tlist}}\vec{M}\,(\mathsf{Tcons}\,b\,bs) \\ &\mapsto M_4\,b\,bs\ \big((\mathcal{R}_{\mathsf{tree}}\vec{M}) \circ b\big)\big((\mathcal{R}_{\mathsf{tlist}}\vec{M}) \circ bs\big) \\ &= M_4\,b\,bs\ (+\,b)(\oplus\ bs) \\ &\to \lambda a.\mathsf{Tcons}(+\,b\,a)(\oplus\ bs\ a). \end{aligned}$$