Inductive types in constructive languages

Bruin, Peter Johan de

Chapter 6

Recursors in constructive type theories

In this chapter we present several styles of introducing recursive functions on an inductive type. The inductive type may be characterized in two ways:

1. as a well-founded relation $(\prec) : \subseteq T^2$, for which we have recursion principle (3.11):

$$\frac{\begin{array}{l} U : \mathbf{Type} \\ s(x : T;\ h : U^{|\prec x|}) : U \end{array}}{\exists! f : U^T :: \forall x : T :: fx = s(x; f)}$$

2. or as an initial F-algebra $(T; \tau)$ (section 4.3), for which we have

$$\frac{\begin{array}{l} U : \mathcal{C} \\ \psi : F.U \to U \end{array}}{(\!|U; \psi|\!) : !\{\, f : T \to U \mid :\, f \circ \tau = \psi \circ F.f \,\}} \ . \tag{6.1}$$

   This is sometimes called the iteration principle, and $(\!|U; \psi|\!)$, or $(\!|\psi|\!)$, is called a catamorphism [57]. If $\mathcal{C}$ is **TYPE**, we have a well-founded predecessor relation $\prec$ as given in section 4.5.

We shall first derive the recursion principle for initial F-algebras, then we derive in 6.2 the construction of dependent functions using either kind of inductive type characterization. In 6.3 we consider Mendler's style of recursion which comes closer in form to the unrestricted form of recursive equation, $f(\tau.y) = E_{f,y}$. Finally we shall see in 6.4 how each style of recursion generalizes to mutual recursion, and how an alternative form of mutual recursion may be useful.

6.1 Algebraic recursion, or paramorphisms

The function $\psi : F.U \to U$ in (6.1) that determines the value $(\!|\psi|\!).x$ of a catamorphism cannot use the predecessors $y : \prec x$ directly but only the values $(\!|\psi|\!).y$. To overcome this, the following recursion principle is derived, which corresponds more closely to (3.11). We formulate name it in the style of Meertens [58].

Theorem 6.1 (Paramorphisms) *An F-algebra $(T;\tau)$ in a category with binary products is initial iff one has*

$$\frac{\begin{array}{l} U\colon\mathcal{C} \\ \psi\colon F.(T\times U)\to U \end{array}}{\exists!\, f\colon T\to U :: f\circ\tau = \psi\circ F.\langle \mathsf{Id}, f\rangle} \tag{6.2}$$

The function f produced by this rule is called a paramorphism, *and is noted $[\![\psi]\!]$.*

Proof. $\Rightarrow$:

$$\begin{array}{lcll} & f\circ\tau = \psi\circ F.\langle \mathsf{Id}, f\rangle & & \\ \Leftrightarrow & \langle \mathsf{Id}, f\rangle\circ\tau = \langle \tau\circ F.\pi_0,\ \psi\rangle\circ F.\langle \mathsf{Id}, f\rangle & \{\text{functor properties}\} \\ \Leftrightarrow & \langle \mathsf{Id}, f\rangle = (\!|T\times U;\ \phi|\!) & \{(6.1),\ \text{defining } \phi := \langle \tau\circ F.\pi_0, \psi\rangle\} \\ \Leftrightarrow & f = \pi_1\circ(\!|\phi|\!) & \{\text{as } \mathsf{Id} = \pi_0\circ(\!|\phi|\!)\} \end{array}$$

So we have:

$$[\![\psi]\!] \ :=\ \pi_1\circ(\!|\phi|\!) \ \underline{\text{where}}\ \phi := \langle \tau\circ F.\pi_0, \psi\rangle\ . \tag{6.3}$$

$\Leftarrow$: Assume (6.2), and $\psi\colon F.U\to U$. We seek to find a unique homomorphism:

$$\begin{array}{lcll} & f\circ\tau = \psi\circ F.f & & \\ \Leftrightarrow & f\circ\tau = \psi\circ F.(\pi_1\circ\langle \mathsf{Id}, f\rangle) & \{\text{products}\} \\ \Leftrightarrow & f\circ\tau = (\psi\circ F.\pi_1)\circ F.\langle \mathsf{Id}, f\rangle & \{\text{functor}\} \\ \Leftrightarrow & f = [\![\psi\circ F.\pi_1]\!] & \{\ (6.2)\ \} \end{array}$$

■

See section 7.1 for dual catamorphisms called *anamorphisms*. Malcolm [52] and Fokkinga [30] give these and some more schemes of recursive functions with names like *zygomorphisms*, *mutumorphisms*, *prepromorphisms*, and *postpromorphisms*.

Example 6.1 For natural numbers, with $F.X := 1 + X$ and $[\mathsf{K}\,0, \lambda\,\mathsf{s}] = \tau$, one can get the usual recursor $R_U\colon U\to(\mathbb{N}\to U\to U)\to\mathbb{N}\to U$ of typed lambda calculus, that satisfies

$$R_U a g 0 = a \qquad R_U a g (\mathsf{s}\,n) = g n (R_U a g n)\ ,$$

by taking for $R_U a g$ the paramorphism $[\![\psi]\!]$ where

$$\psi := [\mathsf{K}\,a, ((n,u)\mapsto g n u)]:\ \ 1 + T\times U\to U\ .$$

(End of example)

It should be noted that, though $[\![\psi]\!]$ as defined by (6.3) satisfies the equation given by (6.2), the reduction rule that we actually get is somewhat different:

$$[\![\psi]\!]\circ\tau \ =>\ \pi_1\circ(\!|\phi|\!)\circ\tau \ =>\ \pi_1\circ\phi\circ F.(\!|\phi|\!) \ =>\ \psi\circ F.(\!|\phi|\!)$$

Proof. $\Rightarrow$: assume that $(T;\tau)$ is initial, and that the rule premises hold. To get the dependent function f, we seek some $f'\!:T \to \Sigma(T;U)$ with $\mathsf{fst}(f'.x) = x$, so that we can take $fx := \mathsf{snd}(f'.x)$. We derive f' from the specification of f as follows.

$$
\begin{array}{lcll}
 & \lambda\mathsf{fst} \circ f' = \mathsf{I} \ \wedge\ \forall(y :: \mathsf{snd}(f'.(\tau.y)) = s(F.f'.y)\,) & & \\
\Leftrightarrow & \lambda\mathsf{fst} \circ f' \circ \tau = \tau \circ F.(\lambda\mathsf{fst} \circ f') \ \wedge\ \forall(y :: \mathsf{snd}(f'.(\tau.y)) = s(F.f'.y)\,) & \{\text{th. 4.1}\} \\
\Leftrightarrow & \forall y :: f'.(\tau.y) = (\tau.(F.\lambda\mathsf{fst}.(F.f'.y)); s(F.f'.y)) & & \\
\Leftrightarrow & f' \circ \tau = (h \mapsto (\tau.(F.\lambda\mathsf{fst}.h); s(h))) \circ F.f' & & \\
\Leftrightarrow & f' = (\![h \mapsto (\tau.(F.\lambda\mathsf{fst}.h); s(h))]\!) & \{\text{catamorphism}\} &
\end{array}
$$

(This proves also that f', and hence f, is unique.)

$\Leftarrow$: assuming (6.5), we prove that $(T;\tau)$ is initial by deriving the paramorphism rule (6.2). Given some $U\!:\mathbf{Type}$ and $\psi\!:F.(T \times U) \to U$, apply (6.5) to $Ux := U$; $s(h) := \psi.(F.((z;u) \mapsto (z,u)).h)$. Say this yields $f'\!:\Pi(T;U)$, then take $f.x := f'x$, which clearly satisfies the requirement $f \circ \tau = \psi \circ F.\langle \mathsf{I}, f\rangle$.

It remains to check that this f is unique. So, assuming that some $g\!:T \to U$ satisfies $g \circ \tau = \psi \circ F.\langle \mathsf{I}, g\rangle$ too, we prove that f and g are equal:

$$
\begin{array}{lcll}
 & f = g & & \\
\Leftrightarrow & \forall x\!:T :: f.x = g.x & & \\
\Leftrightarrow & \exists\,\Pi(T;U') & \underline{\text{where}}\ U'x := (f.x = g.x) & \\
\Leftarrow & \exists\,\Pi(h\!:F.\Sigma(T;U') :: U'(\tau.(F.\lambda\mathsf{fst}.h)) & \{(6.5)\} & \\
\Leftarrow & \forall h\!:F.\Sigma(T;U') :: f.(\tau.(F.\lambda\mathsf{fst}.h)) = g.(\tau.(F.\lambda\mathsf{fst}.h)) & & \\
\Leftrightarrow & \psi \circ F.(\langle \mathsf{I}, f\rangle \circ \lambda\mathsf{fst}) = \psi \circ F.(\langle \mathsf{I}, g\rangle \circ \lambda\mathsf{fst}) :\ F.\Sigma(T;U') \to T & \{\text{property } f \text{ and } g\} & \\
\Leftarrow & \langle \mathsf{I}, f\rangle \circ \lambda\mathsf{fst} = \langle \mathsf{I}, g\rangle \circ \lambda\mathsf{fst} :\ \Sigma(T;U') \to T & & \\
\Leftrightarrow & \forall x\!:T;\ f.x = g.x :: (x, f.x) = (x, g.x) & & \\
\Leftrightarrow & \mathsf{True} & &
\end{array}
$$

■

As the $f\!:\Pi(T;U)$ in (6.5) is unique, we can give it a name: $\mu\_\mathsf{rec}(U;s)$. We remark that the proof for the '$\Leftarrow$'-part in theorem 6.3 contains *two* abstract applications of (6.5), one to construct a paramorphism and one to prove that it is unique.

Most typical examples of dependent recursion arise from inductive proofs: given a property $P\!:\mathbf{Prop}^T$ and an inductive proof of $\forall x :: Px$, the proof object (or tree) corresponding to this proof is given by a dependent recursion. A simple concrete example is the following.

Example 6.2 Consider the natural numbers as an initial $(\mathsf{K}\,1{+}\mathsf{I})$-algebra, $(\mathbb{N};[\mathsf{K}\,0,\lambda\mathsf{s}])$. We construct, for any $n\!:\mathbb{N}$, the function $f_n\!:\mathbb{N}^2 \to \mathbb{N}^n$ that transforms (a,b) into the n-tuple $(a, a\cdot b, \ldots, a\cdot b^{n-1})$. The recursion equations are

$$
\begin{array}{rcl}
f_0 & = & (a,b) \mapsto () \\
f_{\mathsf{s}\,n} & = & (a,b) \mapsto (a, f_n.(a\cdot b, b))
\end{array}
$$

Now, equation (6.5) says that for any s of appropriate type, there exists an f such that $f_0 = s(0;0)$, $f_{\mathsf{S}\,n} = s(1;n;f_n)$. So we just have to take

$$\begin{aligned} U_n &:= \mathbb{N}^n \\ s(0;0) &:= (a,b) \mapsto () \\ s(1;n;f') &:= (a,b) \mapsto (a, f'.(a \cdot b, b)) \end{aligned}$$

and obtain an $f := \mu\_\mathsf{rec}(U;s)$ that satisfies our recursion equations.

6.2.3 Dependent recursion in Paulin style. The second possibility is to keep $y{:}\,F.T$ separate from the tuple of values fz. This is done in most languages with dependent recursion, where the inductive type is usually defined by a finite set of production rules. A formulation based on a functor F seems only possible for polynomial F, and requires us to extend F to operate on families of types and on dependent functions. This was done by Coquand and Paulin in [22], as follows.

Let $F.X = \Sigma(a{:}\,A :: X^{Ba})$. We extend F to operate on families $U{:}\,\mathbf{Type}^T$ and on dependent functions $f{:}\,\Pi(T;U)$, in such a way that:

$$\frac{U{:}\,\mathbf{Type}^T}{F'.U{:}\,\mathbf{Type}^{F.T}} \qquad \frac{f{:}\,\Pi(T;U)}{F.f{:}\,\Pi(F.T;F'.U)}$$

For $y{:}\,F.T$, $(F.f)y$ has to be the tuple of function values fz for all components $z{:}\,T$ of y, and $(F'.U)y$ is the type of this tuple. Thus:

$$\begin{aligned} (F'.U)(a;t) &:= \Pi(y{:}\,Ba :: U(t_y)) \\ (F.f)(a;t) &:= (y :: f(t_y)) \end{aligned}$$

You may note that $F.\Sigma(T;U) \cong \Sigma(F.T;F'.U)$. Now, the rule becomes (we leave the proof to the reader):

$$\frac{\begin{array}{l} U{:}\,\mathbf{Type}^T \\ s(y{:}\,F.T;\ h{:}\,(F'.U)y){:}\,U(\tau.y) \end{array}}{\exists f{:}\,\Pi(T;U) :: \forall y{:}\,F.T :: f(\tau.y) = s(y;(F.f)y)} \tag{6.6}$$

6.3 Mendler's approach

Mendler [59] introduces a somewhat different style of recursion over an initial F-algebra μF. The idea is here that in order to define a (dependent) function $f{:}\,\Pi(\mu F;U)$, one may assume that the function is already available on some subset $X{:}\subseteq \mu F$ while defining it on $F.X$. This gives a recursive equation for f that is simpler than the one appearing in (6.5). Mendler's thesis [59] uses a distinguished inclusion relation on types that is defined by separate production rules, and which we note $(\subseteq_\mathsf{m}){:}\subseteq \mathbf{Type}^2$. The rule looks like:

$$\frac{\begin{array}{l} U{:}\,\mathbf{Type}^T; \\ X{:}\,\mathbf{Type};\ X \subseteq_\mathsf{m} T;\ h{:}\,\Pi(X;U) \vdash s(h){:}\,\Pi(y{:}\,F.X :: U(\tau.y)) \end{array}}{\exists f{:}\,\Pi(T;U) :: \forall y{:}\,F.T :: f(\tau.y) = sfy} \tag{6.7}$$

Note that, as in (6.4) and (6.5), rule (6.7) does not need to state that the constructed f is unique, for this can be derived by employing the dependency in the type of U, again taking $U'x := (f.x = g.x)$.

Example 6.3 Rule (6.7) yields the recursion equations of example 6.2 when we define s simply by:

$$\begin{array}{lcl} sf(0;0) & := & (a,b) \mapsto () \\ sf(1;n) & := & (a,b) \mapsto (a, f(n).(a \cdot b, b)) \end{array}$$

(End of example)

Derivation of (6.7) requires a semantical analysis of the predicate $\subseteq_{\mathsf{m}}$, which we will not do here. But in his paper [60] Mendler replaced the inclusion relation $X \subseteq_{\mathsf{m}} T$ by an explicit function $i\colon X \to T$, and used only non-dependent functions. Correctness of this principle requires that the polymorphic dependency on X, h, and i be uniform in a certain way. This is covered by the *naturality* principle, described in appendix D for languages without dependent types. The resulting rule holds in any category $\mathcal{C}$ with binary products. Thus we get:

$$\frac{\begin{array}{l} U\colon \mathcal{C}; \\ X\colon \mathcal{C};\ i\colon X \to T;\ h\colon X \to U \vdash s_X(i,h)\colon F.X \to U \\ \text{where } s \text{ is } \textit{natural} \end{array}}{\exists! f\colon T \to U :: f \circ \tau = s_T(\mathsf{Id}, f)} \tag{6.8}$$

where 's is natural' means that, for all $p\colon X \to X'$; $i'\colon X' \to T$; $h'\colon X' \to U$, one has

$$s_X(i' \circ p,\ h' \circ p) \ = \ s_{X'}(i', h') \circ F.p \ .$$

As indicated in appendix D, any lambda-definable s is natural. Therefore, this requirement can be omitted in calculi where one has only lambda-definable objects.

Example 6.4 We construct a non-dependent variant of the function of example 6.2, namely $f\colon \mathbb{N} \to \mathbb{N}^2 \to \mathsf{Clist}\,\mathbb{N}$ satisfying

$$\begin{array}{rcl} f.0 & = & (a,b) \mapsto \Box \\ f.\mathsf{s}\,n & = & (a,b) \mapsto a +\!\!< f.(a \cdot b, b) \end{array}$$

This function is produced by (6.8) when we take for s:

$$\begin{array}{lcl} s_X(i,h).(0;0) & := & (a,b) \mapsto \Box \\ s_X(i,h).(1;x) & := & (a,b) \mapsto a +\!\!< h.(a \cdot b, b) \end{array}$$

Theorem 6.4 *An F-algebra $(T;\tau)$ in any category with binary products is initial iff it satisfies (6.8).*

Proof. $\Rightarrow$: Let $(T;\tau)$ be initial, and assume an s that satisfies the premises of (6.8). We calculate the unique solution for f by showing that $\langle \mathsf{Id}, f\rangle$ is a homomorphism, as follows.

$$
\begin{array}{lcll}
 & f\circ\tau = s_T(\mathsf{Id}, f) & \\
\Leftrightarrow & f\circ\tau = s_T(\pi_0\circ\langle \mathsf{Id}, f\rangle,\ \pi_1\circ\langle \mathsf{Id}, f\rangle) & \{\text{products}\} \\
\Leftrightarrow & f\circ\tau = s_{T\times U}(\pi_0,\pi_1)\circ F.\langle \mathsf{Id}, f\rangle & \{s \text{ is natural}\} \\
\Leftrightarrow & \langle \tau,\ f\circ\tau\rangle = \langle \tau,\ s(\pi_0,\pi_1)\circ F.\langle \mathsf{Id}, f\rangle\rangle & \{\text{products}\} \\
\Leftrightarrow & \langle \mathsf{Id}, f\rangle\circ\tau = \langle \tau\circ F.\pi_0,\ s(\pi_0,\pi_1)\rangle\circ F.\langle \mathsf{Id}, f\rangle & \{\text{products, } F \text{ a functor}\} \\
\Leftrightarrow & \langle \mathsf{Id}, f\rangle = (\![T\times U;\ \langle \tau\circ F.\pi_0,\ s(\pi_0,\pi_1)\rangle]\!) & \{\text{initiality}\} \\
\Leftrightarrow & f = \pi_1\circ(\![T\times U;\ \langle \tau\circ F.\pi_0,\ s(\pi_0,\pi_1)\rangle]\!) & \{\text{fact below}\}
\end{array}
$$

Writing $\phi := \langle \tau\circ F.\pi_0,\ s(\pi_0,\pi_1)\rangle$, we used the fact:

$$
\begin{array}{lcll}
 & \mathsf{Id} = \pi_0\circ(\!|\phi|\!) & \\
\Leftrightarrow & \pi_0\circ(\!|\phi|\!)\circ\tau = \tau\circ F.(\pi_0\circ(\!|\phi|\!)) & \{\text{theorem 4.1}\} \\
\Leftrightarrow & \pi_0\circ\phi\circ F.(\!|\phi|\!) = \tau\circ F.(\pi_0\circ(\!|\phi|\!)) & \{\text{catamorphism}\} \\
\Leftrightarrow & \tau\circ F.\pi_0\circ F.(\!|\phi|\!) = \tau\circ F.(\pi_0\circ(\!|\phi|\!)) & \{\text{definition } \phi\} \\
\Leftrightarrow & \mathsf{True} & \{F \text{ a functor}\}
\end{array}
$$

$\Leftarrow$: Simple; given $\phi\colon F.U\to U$, apply (6.8) to $s_X(i,h) := \phi\circ F.h$ which is obviously natural. ■

As with paramorphisms, the actual reduction rule that we get when f is defined as $\pi_1\circ(\!|\phi|\!)$ is not $f\circ\tau \Rightarrow s(\mathsf{Id}, f)$, but rather:

$$f\circ\tau \ \Rightarrow\ s(\pi_0,\pi_1)\circ F.(\!|\phi|\!)\ .$$

A Mendler rule for dependent functions that uses an explicit inclusion function can be given, but it appears to be too complicated to be practical:

$$
\frac{\begin{array}{l} U\colon \mathbf{Type}^T; \\ X\colon \mathbf{Type};\ i\colon X\to T;\ h\colon \Pi(x\colon X :: U(i.x)) \vdash s_X(i,h)\colon \Pi(y\colon F.X :: U(\tau.(F.i.y))) \\ \text{where } s \text{ is natural} \end{array}}{\exists f\colon \Pi(T;U) :: \forall y\colon F.T :: f(\tau.y) = s_T(\mathsf{I}, f)y} \qquad (6.9)
$$

It is probably not possible to derive rule (6.7) directly, because of the special role of the inclusion relation. Rather, one would have to prove that any construction made under an inclusion assumption $X \subseteq_{\mathsf{m}} T$ can be transformed into one using a function $i\colon X\to T$. We will not try to do so.

Parameter h in premise s in rules (6.7) and (6.8) gives access to the function value on immediate predecessors $x\colon X$ of the function argument $\tau.y$. Either rule can be strengthened to allow access to the function value on non-immediate predecessors. For (6.7), this

is done by adding a hypothesis $X \subseteq_{\mathsf{m}} F.X$, for (6.8) by adding a parameter $d\colon X \to F.X$. Assuming that $\tau^{\cup}\colon T \to F.T$ is available, the latter rule becomes:

$$\frac{\begin{array}{l} U\colon \mathcal{C};\\ X\colon \mathcal{C};\ i\colon X \to T;\ d\colon X \to F.X;\ h\colon X \to U \vdash s_X(i,d,h)\colon F.X \to U\\ \text{where } s \text{ is natural}\end{array}}{\exists!\, f\colon T \to U :: f \circ \tau = s_T(\mathsf{Id}, \tau^{\cup}, f)} \tag{6.10}$$

To derive this rule, one has to instantiate X not to $T \times U$, but to some type that encodes the function value on all predecessors of $y\colon F.X$. An initial $(F \times \mathsf{K}\, U)$-algebra, say $(V\colon \mathcal{C};\ \kappa\colon F.V \times U \to V)$, would suit well, for then we can instantiate

$$\begin{array}{rcl} i\colon V \to T & := & (\!|\pi_0 \mathbin{\bar{\circ}} \tau|\!)\\ d\colon V \to F.V & := & \kappa^{\cup} \mathbin{\bar{\circ}} \pi_0\\ h\colon V \to U & := & \kappa^{\cup} \mathbin{\bar{\circ}} \pi_1 \end{array}$$

Further proof details are left to the reader.

6.4 Recursors for mutual induction and recursion

Considering mutual recursion, we have to distinguish between mutually recursive functions on a single inductive type and recursive functions on a family of mutually inductive types.

6.4.1 Mutual recursion on a single inductive type. Regarding the first kind of mutual recursion, note that a tuple of functions on a single inductive type, e.g. $f_0\colon T \to B_0$, $f_1\colon T \to B_1$, is equivalent to a single function with a cartesian product as codomain, $f\colon T \to B_0 \times B_1$. Therefore, in a calculus that has cartesian products (finite or infinite), any recursion principle can be employed to construct mutually recursive functions.

6.4.2 Standard recursion on mutually inductive types. We modeled mutually inductive types (section 5.2) by several forms of initial algebras in an exponential category. All categorical recursion principles for initial algebras that we presented: (6.1), (6.2), and (6.7), can be interpreted in these categories, yielding arrows $f\colon T \to U$ <u>in</u> $\mathbf{TYPE}^N$. The recursors for dependent functions, (6.5) and (6.7), can easily be accommodated in an exponential category too; for example, rule (6.5) becomes

$$\frac{\begin{array}{l} U_{i:N}\colon \mathbf{Type}^{Tn};\\ s_{i:N}(h\colon F_i.\Sigma(T;U))\colon U_i(\tau_i.(F_i.\lambda\mathsf{fst}^N.h))\end{array}}{\exists f\colon \Pi(N;\Pi(T;U)) :: \forall i\colon N;\ y\colon F_i.T :: f_i(\tau_i.y) = s_i(F_i.(n' :: z \mapsto (z; f_i z)).y)}$$

where Σ en Π have to be lifted: $\Pi(T;U)_i := \Pi(T_i;U_i)$.

6.4.3 Liberal mutual recursion. Standard recursion on a family of N inductive types above requires that the recursive functions f consist of one function f_i for each type T_i. As an alternative recursion scheme, it is sometimes more convenient to index the functions over some type M that is different from N, and use a mapping $d(j\colon M)\colon N$ to indicate the domain of function f_j. The defining equation for $f_j.(\tau_{dm}.y)$ may assume that, for every predecessor $x\colon T_i$ of y, the function results $f_{m'}.x$ for each $m'\colon M$ with $dm' = n$ are available. We name the type of this tuple of function results $U|_{=n}$, so $U|_=$ is the tuple of all these types. We dub the rule "liberal mutual recursion", as the function index type M is not fixed to be the index type N.

Theorem 6.5 (Liberal mutual recursion) *For any endofunctor F on $\mathbf{TYPE}^N$, an F-algebra $(T\colon \mathbf{TYPE}^N; \tau\colon F.T \to T)$ is initial iff rule (6.11) below holds for any $M\colon \mathbf{Type}$; $d\colon N^M$; $U\colon \mathbf{Type}^M$. We abbreviate:*

$$\begin{array}{rcl}
U|_= : \mathbf{Type}^N & := & (n :: \Pi(\{j\colon M \mid\colon dm = n\};\, U)); \\
T_d\colon \mathbf{Type}^M & := & (m :: T_{(dm)}) \\
F_d\colon \mathbf{TYPE}^M \to \mathbf{TYPE}^M & := & S \mapsto (m :: F_{(dm)}.S) \\
\tau_d\colon F_d.T_d \to T_d \ \underline{\mathsf{in}}\ \mathbf{TYPE}^M & := & (m :: \tau_{(dm)}) \\
f\colon T_d \to U;\ i\colon N \vdash \quad f|_{=n}\colon T_i \to U|_{=n} & := & x \mapsto (m :: f_j.x)
\end{array}$$

$$\frac{\psi\colon F_d.(T \times U|_=\ ) \to U \ \underline{\mathsf{in}}\ \mathbf{TYPE}^M}{\exists! f\colon T_d \to U :: f \circ \tau_d = \psi \circ F_d.\langle \mathsf{Id}, f|_=\ \rangle)} \tag{6.11}$$

Proof. $\Rightarrow$: We calculate the unique f that satisfies the specification, by translating it into an equation in category $\mathbf{TYPE}_N$:

$$\begin{array}{lcll}
 & \forall m :: f_j \circ \tau_{dm} = \psi_j \circ F_{dm}.\langle \mathsf{Id}, f|_=\ \rangle & & \\
\Leftrightarrow & \forall n;\, m;\, dm = n :: f_j \circ \tau_i = \psi_j \circ F_i.\langle \mathsf{Id}, f|_=\ \rangle & \{\text{introduce } n = dm\} \\
\Leftrightarrow & \forall n :: f|_{=n} \circ \tau_i = \psi|_{=n} \circ F_i.\langle \mathsf{Id}, f|_=\ \rangle & \{\text{definition } |_{=n}\} \\
\Leftrightarrow & f|_= \ = [\![\psi|_=\ ]\!] & \{(6.2)\} \\
\Leftrightarrow & \forall m :: f_j = x \mapsto ([\![\psi|_=\ ]\!]_{dm}.x)_j &
\end{array}$$

$\Leftarrow$: This rule subsumes the paramorphism principle (6.2) with $\mathcal{C} := \mathbf{TYPE}^N$, by instantiating $M := N$, $dm := m$. ∎

The rule of liberal mutual recursion made use of the equality type. However, in a calculus without explicit equality, one might still allow restricted forms of this rule by using syntactic checks for the equality $dm = n$, as the following example illustrates.

Example 6.5 Suppose we have a family of inductive types $T\colon \mathbf{Type}^{\mathbb{N}\times\mathbb{N}}$, and we wish to simultaneously define two families of recursive functions,

$$\begin{array}{c}
g_n\colon T_{nn} \to \mathbb{N} \\
h_{nm}\colon T_{nm} \to T_{mn}
\end{array}$$

This is possible by rule (6.11), taking

$$\begin{aligned} M &:= \mathbb{N} + \mathbb{N}^2 \\ d(0;n) &:= (n,n) \\ d(1;n,m) &:= (n,m) \\ U(0;n) &:= \mathbb{N} \\ U(1;n,m) &:= T_{mn} \end{aligned}$$

After selecting a suitable ψ, the rule yields an f from which we can obtain $g_n := f_{(0;n)}$ and $h_{nm} := f_{(1;n,m)}$. Using the currying convention of subsection 2.12.5, we can write $(g,h) = f$. The characteristic equations become

$$\begin{aligned} g_n \circ \tau_{nn} &= \psi_{0n} \circ F_{0n}.\langle \mathsf{Id}, (g,h)|_{=} \rangle \\ h_{nm} \circ \tau_{nm} &= \psi_{1nm} \circ F_{1nm}.\langle \mathsf{Id}, (g,h)|_{=} \rangle \end{aligned}$$

Inspection of the right-hand side of these equations reveals that the expressions which define $g_n.(\tau_{nn}.y)$ and $h_{nm}.(\tau_{nm}.y)$ may contain reference to y, to $h_{n'm'}.z$ for any immediate predecessor $z{:}\,T_{n'm'}$ of $\tau_{nm}.y$, and also to $g_{n'}.z$ when it happens that $m' = n'$. If one allows the latter only when m' and n' are equal by definitional equality, no explicit equality predicate is necessary.

6.5 Summary

This chapter completed our expedition of describing ordinary inductive types: chapter 2 introduced our language, chapter 4 our categorical machinery, chapter 5 surveyed schemes for inductive type definitions, and this chapter finished with describing the forms of recursion over an inductive type.

We described the following forms:

1. Catamorphisms (6.1), obtained directly from initiality.
2. Paramorphisms (6.2), which follow the scheme of simple or algebraic recursion.
3. Dependent (algebraic) recursion (6.5) and (6.6)
4. Mendler recursion (6.8), using a quantifiction over types. Any of the recursors 1–3 above can be formulated in Mendler form, giving six combinations.
5. Liberal mutual recursion (6.11). Any of the six combinations above can be generalized to either standard or liberal mutual recursion, giving twelve forms of mutual recursion.

Furthermore, any of these may appear either in a weak form, giving just a typing rule and an equality (or reduction) rule, or in a strong form, giving also a uniqueness condition. The latter is only possible when the calculus has an explicit equality predicate.

The strong forms of the recursors and the weak form of dependent recursion are all equivalent (with respect to extensional equality), with these remarks:

6.2 Recursive dependent functions

We now specialize to the category of types. Usage of dependent types allows the transfinite induction principle (3.6) and transfinite recursion (theorem 3.7) for well-founded relations to be unified into a single dependent recursion principle. We have two similar principles for initial F-algebras, described in 6.2.2 and 6.2.3. In all cases, the principle does not need to state that the constructed function f is unique, for this is derivable by an auxiliary application of the very same principle!

6.2.1 Dependent recursion over a well-founded relation.

Theorem 6.2 *A relation $(\prec)\colon\subseteq T^2$ on a type T is well-founded iff one has:*

$$\frac{\begin{array}{l} U\colon \mathbf{Type}^T; \\ s(x\colon T;\ h\colon (z\colon \prec x \triangleright Uz))\colon Ux \end{array}}{\exists f\colon \Pi(T;U) :: \forall x\colon T :: fx = s(x; f|_{\prec x})} \tag{6.4}$$

Proof. $\Rightarrow$: This runs parallel to theorem 3.7. First we inductively define a subset $R\colon\subseteq \Sigma(T;U)$ by

$$\forall x\colon T;\ h\colon (z\colon \prec x \triangleright Uz)\ ::\ \forall (z\colon \prec x :: (z;hz) \in R) \Rightarrow (x; s(x;h)) \in R\ .$$

Then one proves by transfinite induction (3.6) that R is single-valued in the sense that

$$\forall x\colon T :: \exists! u\colon Ux :: (x;u) \in R\ ,$$

which proof we skip here. Letting p be the corresponding proof term, we can take $fx := \iota(px)$.

$\Leftarrow$: Rule (6.4) subsumes (3.6) by taking $Ux := Px$. Note also that it subsumes theorem 3.7 by substituting $Ux := U$. ∎

6.2.2 Dependent recursion on an initial F-algebra. To formulate a rule for dependent recursion on an initial F-algebra, one has to find a way to encode the hypothesis of the induction step. This hypothesis should contain, for some $y\colon F.T$, for each predecessor $z\colon T$ of $\tau.y$ the function value $fz\colon Uz$. One possibility is to replace each predecessor z by the pair $(z;fz)\colon \Sigma(T;U)$, so the hypothesis becomes

$$h\colon F.\Sigma(T;U)\ .$$

A second possibility, which we shall consider in 6.2.3, is to add to y the tuple of all these function values fz. Pursuing the first possibility, we get the following.

Theorem 6.3 *An F-algebra $(T;\tau)$ in the category of types is initial iff the following rule holds.*

$$\frac{\begin{array}{l} U\colon \mathbf{Type}^T; \\ s(h\colon F.\Sigma(T;U))\colon U(\tau.(F.\lambda\mathsf{fst}.h)) \end{array}}{\exists f\colon \Pi(T;U) :: \forall y\colon F.T :: f(\tau.y) = s(F.(z \mapsto (z; fz)).y)} \tag{6.5}$$

- Equality types are required in order to formulate and derive the general form of liberal mutual recursion.
- Equality types are also required to derive any strong recursor from weak dependent recursion.
- Generalized sums are required to derive strong or weak dependent recursion.

"Equivalent" means here, that any application of one recursor can be translated into an application of the other recursor that satisfies the same equation. However, the actual reduction behavior may differ.

Similarly, the weak forms of the catamorphism, paramorphism, Mendler, and liberal mutual recursion rules are equivalent, with the same remarks.