Inductive types in constructive languages

Bruin, Peter Johan de

Chapter 3

Common induction and recursion principles

In this chapter, we present a summary of some common principles of inductive definition and recursion. This serves two purposes: to get introduced to inductive definitions and also to get acquainted with our notations.

In 3.1, we start with some simple examples, each one adding a new aspect of inductive definition: naturals, lists, trees, join lists, rose trees, ordinal notations, inductive relations, and infinite lists.

In 3.2, we formulate the main derived principles of induction and recursion on naturals.

In 3.3, we summarize the theory of inductively defining a subset of a given set.

In 3.4, we derive some recursion principles from the induction rules in 3.3.

3.1 Examples of inductive types

Before embarking on generalized formulations of induction principles, we list some examples of inductively defined types that illustrate several features.

Example 3.1 (Natural numbers) The type $\mathbb{N}$ is usually described by the five *Peano axioms* (where we ignore the presence of $\mathbb{N}$ in *ADAM*):

1. 0, *zero*, is a natural number: $0 : \mathbb{N}$.
2. Whenever n is a natural number, then $\mathsf{s}\, n$, its *successor*, is also a natural number:
$$n : \mathbb{N} \vdash \mathsf{s}\, n : \mathbb{N}$$
3. No successor equals zero: $\mathsf{s}\, n \neq 0$.
4. Natural numbers with the same successor are equal: $\mathsf{s}\, n = \mathsf{s}\, m \Rightarrow n = m$.
5. Nothing else is a natural number. That is, if a predicate $P(n : \mathbb{N})$ holds for zero and is preserved by the successor operation, then it holds for all natural numbers:
$$P(0) \wedge \forall (n : \mathbb{N} :: P(n) \Rightarrow P(\mathsf{s}\, n)) \;\Rightarrow\; \forall (n : \mathbb{N} :: P(n)) \tag{3.1}$$

Regarding this definition, we distinguish a base clause 1, a step clause 2, equality rules 3 and 4, and an induction clause 5. The induction clause is an instance of the elimination principle for naturals (2.3), taking $Tn := Pn$.

Example 3.2 (Lists) For any type A, the type $\mathsf{Clist}\,A$ of *cons lists* is generated by:

$$\begin{array}{rcl} & \vdash & \Box\colon \mathsf{Clist}\,A \\ e\colon A;\ l\colon \mathsf{Clist}\,A & \vdash & e +\!\!< l\colon \mathsf{Clist}\,A \end{array}$$

As for naturals, we have to give clauses saying that two lists are only equal if they are constructed by the same constructor from equal arguments (*no confusion*):

$$\forall(e; l :: e +\!\!< l \neq \Box) \qquad \forall(e, e'; l, l' :: e +\!\!< l = e' +\!\!< l' \Rightarrow e = e' \wedge l = l')$$

and an induction clause saying that all lists can be built by repeated application of the construction rules (*no junk*): for predicates $P(l\colon \mathsf{Clist}\,A)$,

$$P\Box \wedge \forall(e; l :: Pl \Rightarrow P(e +\!\!< l)) \;\Rightarrow\; \forall(l :: Pl)\ .$$

Example 3.3 (Binary trees) The type $\mathsf{BTree}\,A$ is generated by:

$$\begin{array}{rcl} & \vdash & \Box\colon \mathsf{BTree}\,A \\ x\colon A & \vdash & \diamond.x\colon \mathsf{BTree}\,A \\ s, t\colon \mathsf{BTree}\,A & \vdash & s +\!\!+ t\colon \mathsf{BTree}\,A \end{array}$$

where $\Box$ is a unit of $+\!\!+$:

$$s\colon \mathsf{BTree}\,A \;\vdash\; \Box +\!\!+ s = s = s +\!\!+ \Box$$

No other trees are identical (no confusion), i.e.:

$$\begin{array}{rcl} s +\!\!+ t = \Box & \Rightarrow & s = \Box \wedge t = \Box \\ s +\!\!+ t = u +\!\!+ v & \Rightarrow & (s = u \wedge t = v) \vee (s = \Box \wedge t = u +\!\!+ v) \vee (s = u +\!\!+ v \wedge t = \Box) \end{array}$$

Finally, nothing else is a tree (no junk). That is, for predicates $P(t\colon \mathsf{BTree}\,A)$, if

$$P\Box \wedge \forall(x\colon A :: P(\diamond.x)) \wedge \forall(s, t\colon \mathsf{BTree}\,A :: Ps \wedge Pt \Rightarrow P(s +\!\!+ t))$$

then $\forall(s\colon \mathsf{BTree}\,A :: Ps)$.

We will see in example 4.5 that the notion of initial algebra avoids the formulation of complicated no-confusion and no-junk conditions.

Example 3.4 (Join lists) The constructors for join lists $\mathsf{JList}\,A$ are the same as those for $\mathsf{BTree}\,A$, but have the additional equation telling that $+\!\!+$ (called *join*) is associative,

$$s, t, u\colon \mathsf{JList}\,A \;\vdash\; (s +\!\!+ t) +\!\!+ u = s +\!\!+ (t +\!\!+ u).$$

We shall not spell out the complicated no-confusion condition; the no-junk condition is the same as for binary trees.

Example 3.5 (Rose trees) Rose trees and forests (e.g. Malcolm [52]) can be described either as two mutually inductive types or as a single one using lists.

Making the latter choice, rose trees over some type A are inductively generated by: any list f of rose trees together with some element x of A makes a rose tree noted $x \surd f$, i.e.

$$x\colon A, f\colon \mathsf{Clist}(\mathsf{RTree}\, A) \quad \vdash \quad x \surd f\colon \mathsf{RTree}\, A$$

This is an *iterated inductive definition* in the sense of [55], as it builds upon another inductively defined type, Clist. However, the use of iterated induction is not essential here.

The alternative is to define rose trees and lists of them (named forests) simultaneously, by mutual induction:

$$\begin{aligned} x\colon A, f\colon \mathsf{Forest}\, A \quad &\vdash \quad x \surd f\colon \mathsf{RTree}\, A \\ &\vdash \quad \Box\colon \mathsf{Forest}\, A \\ t\colon \mathsf{RTree}\, A, f\colon \mathsf{Forest}\, A \quad &\vdash \quad t +\!\!< f\colon \mathsf{Forest}\, A \end{aligned}$$

Example 3.6 (Ordinal notations) A more typical iterated inductive definition is the following one, which builds upon the inductive type of naturals. Ordinal notations are constructed from zero, a successor operation, and taking the limit of an infinite but countable series of ordinal notations. They may be used to represent ordinals of the so-called second number class. This type Ord is our first infinitary inductive type.

$$\begin{aligned} &\vdash \quad 0\colon \mathsf{Ord} \\ n\colon \mathsf{Ord} \quad &\vdash \quad \mathsf{s}\, n\colon \mathsf{Ord} \\ u\colon \mathsf{Ord}^{\omega} \quad &\vdash \quad \mathsf{lim}\, u\colon \mathsf{Ord} \end{aligned}$$

Our final two examples illustrate rather different kinds of inductive definitions.

Example 3.7 (An inductive relation) Given a relation $R\colon\subseteq T^2$, its transitive closure $R^{(+)}$ is specified by

$$\begin{aligned} R &\subseteq R^{(+)} \\ R^{(+)} \cdot R^{(+)} &\subseteq R^{(+)} \end{aligned}$$

and: $R^{(+)}$ is the least relation w.r.t. $\subseteq$ that satisfies these two rules.

By Knaster-Tarski (theorem 3.6), the unique solution to this specification is the intersection of all relations that satisfy the two rules:

$$R^{(+)} := \bigcap (X\colon\subseteq R \mid\colon R \subseteq X \ \wedge \ X \cdot X \subseteq X)$$

Similarly, the reflexive and transitive closure is

$$R^{(*)} := \bigcap (X\colon\subseteq R \mid\colon |{=_T}| \subseteq X \ \wedge \ R \subseteq X \ \wedge \ X \cdot X \subseteq X)$$

Note the difference with the preceding examples: there we created new types by stating new operations and equations, here we define a subset of an existing type (T^2) by giving rules that refer only to existing operations.

Example 3.8 (Infinite lists) The type $A\infty$ of (total) infinite lists over A is isomorphic with A^ω, but we give a quite different axiomization. This is not an inductive type definition like the examples 3.1–3.7, but we shall see that it is its categorical dual, a final coalgebra instead of an initial algebra. The axioms are:

1. Any infinite list has a head: $\mathsf{hd}\colon A\infty \to A$.
2. Any infinite list has a tail: $\mathsf{tl}\colon A\infty \to A\infty$.
3. For any type X and mappings $h\colon X \to A$ and $t\colon X \to X$, there exists a family of infinite lists $l\colon (A\infty)^X$ such that:
$$\begin{aligned} \mathsf{hd}.l_i &= h.i \ , \\ \mathsf{tl}.l_i &= l_{(t.i)} \ . \end{aligned}$$
4. Moreover, this family of lists l is unique: any other such family equals it.

Summary. Example 3.1 gave the most common form of inductive type definition; example 3.2 defined a type with a parameter; example 3.3 gave a type with parameter and equations; the join lists of example 3.4 added more equations; example 3.5 used mutual induction, example 3.6 iterated induction.

Example 3.7 showed the difference between inductive type and inductive set definitions, and example 3.8 displayed the categorical dual of inductive type definition.

3.2 More on natural numbers

Peano's 5th axiom (3.1) constitutes the first induction principle. Of course one can derive similar principles starting at a base different from zero. A well-known equivalent principle is total induction. Let $|<| := (\lambda\,\mathsf{s})^{(+)}$, the transitive closure of the successor relation $\lambda\,\mathsf{s} = \{n\colon \mathbb{N} :: (n, \mathsf{s}\,n)\}$.

Theorem 3.1 (Total induction) *If a property $P(n\colon \mathbb{N})$ can be proven on the assumption that it holds for smaller natural numbers, then it holds for all natural numbers:*

$$\forall(n\colon \mathbb{N};\ |< n| \subseteq |P| :: Pn) \ \Rightarrow\ \mathbb{N} \subseteq |P| \tag{3.2}$$

Note that $|< n| \subseteq |P|$ abbreviates $\forall(m\colon \mathbb{N} :: m < n \Rightarrow Pm)$.

Proof. Use (3.1) substituting $P(n) := |< n| \subseteq |P|$. ■

Note that no separate treatment of some base case is needed.

Definitions of a recursive function f on natural numbers usually consist of a base case, $f.0 = b$ and an induction step of the form $f.(n+1) = g.(n, f.n)$. This is called primitive recursion. Typed lambda calculi with natural numbers usually have a recursion construct which allows such definitions. With the help of the iota operation, one can derive it from the Peano axioms.

Theorem 3.2 (Primitive recursion)

$$\frac{\begin{array}{l} U\colon \mathbf{Type} \\ b\colon U \\ g\colon \mathbb{N} \times U \to U \end{array}}{\exists!\, f\colon \mathbb{N} \to U :: f.0 = b \ \wedge\ \forall n :: f.\mathsf{s}\, n = g.(n, f.n)} \tag{3.3}$$

Proof. This is an instance of theorem 3.7 below, for $T := \mathbb{N}$, $|\prec| := \lambda \mathsf{s}$, and $s := ((0; h) :: b \mid (\mathsf{s}\, n; h) :: g.(n, hn))$, for Peano's axiom (3.1) says exactly that this $\prec$ is well-founded (3.6). ■

Alternatively, existence of f is a special case of the elimination principle for naturals (2.3) for nondependent types, $Tn := U$. Uniqueness follows: assume $g\colon \mathbb{N} \to U$ satisfies the same equations, then prove $\forall n\colon \mathbb{N} :: g.n = f.n$ by Peano induction, which is an instance of (2.3) too.

3.3 Inductive subset definitions

We summarize the standard theory of inductive set definitions, following the basic definitions from the first section of Aczel's Introduction to Inductive Definitions [3]. This theory deals with subsets of a set (or type) that has been constructed prior to the inductive definition.

3.3.1 Sets inductively defined by rules

Example 3.7 defined $R^{(+)}$ as the least subset of T^2 that is closed under certain rules. The other examples, 3.1 till 3.6, introduced new types, not subsets, but each of these types can be characterized by saying that it equals its own least subset that is closed under certain rules. Let us give the general form of such definitions.

Let type T be given. We define:

1. A *rule* is a pair (X, x) where $X\colon \mathcal{P}T$ is called the set of *premisses* and $x\colon T$ is the *conclusion*. A set of rules, $\Phi\colon \mathcal{P}(\mathcal{P}T \times T)$, is also called a *rule set*.

2. If Φ is a rule set, then a set $S\colon \subseteq T$ is Φ-*closed* iff each rule in Φ whose premisses are in S also has its conclusion in S, i.e. iff

$$\forall (X, y)\colon \in \Phi :: X \subseteq S \Rightarrow y \in S\ .$$

3. If Φ is a rule set, then $I(\Phi)$, the *set inductively defined by* Φ, is given by

$$I(\Phi)\ :=\ \bigcap (S\colon \subseteq T \mid\colon S \text{ is } \Phi\text{-closed})\ . \tag{3.4}$$

Note. Φ-closed sets exists; e.g. the type T itself. Also, the intersection of any collection of Φ-closed sets is Φ-closed. In particular $I(\Phi)$ is Φ-closed and hence $I(\Phi)$ *is the smallest* Φ-*closed subset*.

From the definition (3.4) of $I(\Phi)$ we get immediately the principle of Φ-*induction*: If $P(x\colon T)$ is a predicate, such that whenever $(X, y)\colon \in \Phi$ and $X \subseteq |P|$ then Py, then Px holds for every $x\colon \in I(\Phi)$.

Definition (3.4) involves a second-order quantification. If one designs a logical calculus without quantification over arbitrary subsets, one might consider including inductive set (or predicate) definitions as a primitive rule [55]. By the way, if a calculus includes inductive mutually recursive type definitions in the style of section 5.2.2, then one gets inductive predicate definitions as well.

3.3.2 The well-founded part of a relation

A slightly less general scheme of inductive subset definitions is based on well-founded relations. The constructive idea of a well-founded relation is a generalization of the principle of total induction (3.2), but its classical definition is different.

Let $\prec$ be a binary relation on a type T. The *well-founded part* of T for $\prec$, $W(\prec){:}\subseteq T$, is defined (classically) as the set consisting of those $x{:}T$ for which there is no infinite descending sequence $x \succ s_0 \succ s_1 \succ \cdots$. The relation $\prec$ is called *well-founded* iff $T = W(\prec)$, and it is a *well-ordering* iff it is both well-founded and transitive (but see (3.6) for the constructive definition of wellfoundedness). Note that the transitive closure $\prec^{(+)}$ of any well-founded relation is a well-ordering. The elements $y{:}\prec x$ are called the *predecessors* of x.

The set $W(\prec)$ can be defined inductively using the following rule set $\Phi_\prec$.

$$\Phi_\prec \;:=\; \{x{:}T :: (|\prec x|, x)\} \tag{3.5}$$

The set inductively defined by $\Phi_\prec$, $I(\Phi_\prec)$, is called the *reachable part* of T for $\prec$. The principle of total induction (3.2) can now be formulated as $\mathbb{N} \subseteq I(\Phi_<)$.

Theorem 3.3 *$W(\prec) = I(\Phi_\prec)$, classically.*

Proof. $\supseteq$: It suffices to show that $W(\prec)$ is $\Phi_\prec$-closed. So assume $|\prec x| \subseteq W(\prec)$. To show $x \in W(\prec)$, suppose $x \succ s_0 \succ \cdots$. Then $s_0 \in |\prec x| \subseteq W(\prec)$. But as $s_0 \succ s_1 \succ \cdots$, $s_0 \notin W(\prec)$, which gives a contradiction.

$\subseteq$: Let $x{:}\in W(\prec)$ and S be $\Phi_\prec$-closed. Supposing $x \notin S$, we shall derive a contradiction by finding $x \succ s_0 \succ \cdots$ showing that $x \notin W(\prec)$. As $x \notin S$, then $|\prec x| \not\subseteq S$. Hence there is an $s_0{:}\prec x$ such that $s_0 \notin S$. Repeating indefinitely we obtain $s_{i+1}{:}\prec s_i$ such that $s_{i+1} \notin S$. ∎

Conversely, inductive definitions can often be rephrased in the form $\Phi_\prec$ for a suitable $\prec$. Let Φ be a rule set on a type T; we say Φ is *deterministic* iff

$$(X_0, y) \in \Phi \wedge (X_1, y) \in \Phi \;\Rightarrow\; X_0 = X_1 \;.$$

Let $(\prec_\Phi){:}\subseteq T^2$ be the relation $|\in| \cdot \Phi \;\cup\; \{\,(x,y)\;|{:}\; y \notin \Phi^{>}\}$, i.e.:

$$x \prec_\Phi y \;:=\; y \in \Phi^{>} \Rightarrow \exists(X{:}\mathcal{P}T :: x \in X \wedge (X, y) \in \Phi) \;.$$

(Aczel [3, proposition 1.2.4] erroneously missed the condition $y \in \Phi^{>}$.)

Theorem 3.4 *For deterministic Φ, one has (classically)*

$$I(\Phi) = I(\Phi_{\prec_\Phi}) = W(\prec_\Phi) \;.$$

Proof. We have $I(\Phi_{\prec_\Phi}) = W(\prec_\Phi)$ from theorem 3.3. Next,

$$\begin{array}{lcl}
 & I(\Phi) \subseteq I(\Phi_{\prec_\Phi}) & \\
\Leftarrow & \text{all } \Phi_{\prec_\Phi}\text{-closed sets are } \Phi\text{-closed} & \{\text{def. } I \text{ (3.4)}\} \\
\Leftarrow & \Phi \subseteq \Phi_{\prec_\Phi} & \{\text{def. closedness}\} \\
\Leftrightarrow & \forall (X,y) :\in \Phi :: X = |\prec_\Phi y| & \{\text{def. } \Phi_\prec \text{ (3.5)}\} \\
\Leftrightarrow & \forall (X,y) :\in \Phi :: X = \{\, x \mid : (x,y) \in |\in| \cdot \Phi \} & \{\text{def. } \prec_\Phi,\ y \in \Phi^{>}\} \\
\Leftrightarrow & \Phi \text{ is deterministic} & \{\text{def. deterministic}\}
\end{array}$$

and:

$$\begin{array}{lcl}
 & I(\Phi_{\prec_\Phi}) \subseteq I(\Phi) & \\
\Leftarrow & I(\Phi) \text{ is } \Phi_{\prec_\Phi}\text{-closed} & \{\text{def. } I(\Phi_{\prec_\Phi})\} \\
\Leftrightarrow & \forall y :: |\prec_\Phi y| \subseteq I(\Phi) \Rightarrow y \in I(\Phi) & \{\text{def. } \Phi_\prec, \text{ closedness}\} \\
\Leftarrow & \forall ((X,y) :\in \Phi :: X \subseteq I(\Phi) \Rightarrow y \in I(\Phi)) & \\
 & \quad \wedge \forall (y \notin \Phi^{>} :: T \subseteq I(\Phi) \Rightarrow y \in I(\Phi)) & \{\text{def. } \prec_\Phi\} \\
\Leftrightarrow & \mathsf{True} & \{\text{def. } I(\Phi)\}
\end{array}$$

■

Theorem 3.3 suggests us a constructive interpretation of well-foundedness. Constructively, we even take $T \subseteq I(\Phi_\prec)$ as the definition of well-foundedness. Thus, we henceforth say $\prec$ is well-founded iff it admits *transfinite induction*:

$$\frac{\begin{array}{l} P : \mathbf{Prop}^T \\ \forall y : T;\ |\prec y| \subseteq |P| :: Py \end{array}}{T \subseteq |P|} \tag{3.6}$$

For natural numbers with $(\prec) := (<)$, this is exactly the total induction principle (3.2).

There are several other constructive interpretations of well-foundedness possible, which are classically equivalent:

Theorem 3.5 *The three properties: $\prec$ admitting transfinite induction, $\prec$ having no descending chains, and all nonempty subsets of T having a minimal element, are classically equivalent. Formally these are:*

$$T \subseteq I(\Phi_\prec) \tag{3.7}$$

$$\forall s : T^\omega :: \exists i : \omega :: s_{i+1} \not\prec s_i \tag{3.8}$$

$$\forall Q : \mathcal{P}T;\ \exists (Q) :: \exists q :\in Q :: \neg\exists (Q \cap |\prec q|) \tag{3.9}$$

((3.9) implies (3.8) constructively as well. (3.7) and (3.9) seem to be constructively independent.)

Proof. (3.7) $\Leftrightarrow$ (3.8) is a corollary of theorem 3.3, for (3.8) says just $T \subseteq W(\prec)$.

(3.8) $\Rightarrow$ (3.9): Assume $Q : \mathcal{P}T$, $q :\in Q$ and suppose $\forall q :\in Q :: \exists (Q \cap |\prec q|)$. Then we choose a descending sequence by taking $s_0 := q$, and given $s_i \in Q$, choosing an $s_{i+1} :\in Q \cap |\prec s_i|$. This contradicts (3.8).

(3.9) ⇒ (3.8): Given $s{:}T^\omega$, take $Q := \{i{:}\omega :: s_i\}$. Then by (3.9), $\neg\exists(Q \cap |{\prec}\, s_i|)$ for some i, and in particular $s_{i+1} \not\prec s_i$. ■

A somewhat different use of well-founded relations is to conduct inductive proofs over some given type. To find suitable relations, Paulson [69] described a number of principles to construct well-founded relations.

3.3.3 Inductive definitions as operators

An "operator" (function) $\phi{:}\mathcal{P}T \to \mathcal{P}T$ is *monotonic* iff $(\phi,\phi) \in (\subseteq) \to (\subseteq)$, i.e. iff $X{:}\subseteq Y{:}\mathcal{P}T$ implies $\phi.X \subseteq \phi.Y$. Given ϕ, let rule set Φ_ϕ be defined by

$$\Phi_\phi \;:=\; \{\,(X,y) \mid{:}\; y \in \phi.X\,\}\ .$$

For monotonic ϕ, $S{:}\mathcal{P}T$ is Φ_ϕ-closed just in case $\phi.S \subseteq S$. So

$$I(\Phi_\phi) \;=\; \bigcap(\, S{:}\mathcal{P}T \mid{:}\; \phi.S \subseteq S)\ . \tag{3.10}$$

Hence it is natural to write, still following Aczel [3], $I(\phi)$ for $\bigcap(S{:}\mathcal{P}T \mid{:}\; \phi.S \subseteq S)$.

Conversely, all inductive definitions can be obtained using monotonic operators. For, if Φ is a rule set on T we may define ϕ by

$$\phi.Y \;:=\; \{\, y \mid{:}\; \exists X{:}\subseteq Y :: (X,y) \in \Phi\}\ .$$

Then Y is Φ-closed just in case $\phi.Y \subseteq Y$ so that $I(\Phi) = I(\phi)$.

An alternative characterization of $I(\phi)$ uses transfinite iterations $\phi^{(\lambda)}$ for ordinals λ. We skip this; see [3].

3.3.4 Fixed points in a lattice

The Knaster-Tarski theorem generalizes the fixed point property (3.10) of monotonic operators on sets to complete lattices.

Theorem 3.6 (Knaster-Tarski) *Any monotonic operator F in a complete lattice $(U;\sqsubseteq)$ has a least fixed point*

$$\mathsf{fix}\,F \;:=\; \sqcap(X{:}U \mid{:}\; F.X \sqsubseteq X)$$

and hence by duality a greatest fixed point

$$\sqcup(X{:}U \mid{:}\; X \sqsubseteq F.X)\ .$$

Proof. We have $F.(\mathsf{fix}\,F) \sqsubseteq \mathsf{fix}\,F$ because for any $X{:}U$, if $F.X \sqsubseteq X$ then $\mathsf{fix}\,F \sqsubseteq X$ so $F.(\mathsf{fix}\,F) \sqsubseteq F.X \sqsubseteq X$.

Conversely, $\mathsf{fix}\,F \sqsubseteq F.(\mathsf{fix}\,F)$ follows from $F.(F.(\mathsf{fix}\,F)) \sqsubseteq F.(\mathsf{fix}\,F)$, which holds by monotonicity of F. ■

We show the same proof in linear form:

$$\begin{array}{lcll} & F.(\mathsf{fix}\,F) = \mathsf{fix}\,F & \\ \Leftrightarrow & \mathsf{fix}\,F \sqsubseteq F.(\mathsf{fix}\,F) \ \wedge \ F.(\mathsf{fix}\,F) \sqsubseteq \mathsf{fix}\,F & \{\text{lattice}\} \\ \Leftarrow & F.(F.(\mathsf{fix}\,F)) \sqsubseteq F.(\mathsf{fix}\,F) \ \wedge \ F.(\mathsf{fix}\,F) \sqsubseteq \mathsf{fix}\,F & \{\text{def. fix}\} \\ \Leftrightarrow & F.(\mathsf{fix}\,F) \sqsubseteq \mathsf{fix}\,F & \{F \text{ monotonic}\} \\ \Leftrightarrow & \forall X;\ F.X \sqsubseteq X :: F.(\mathsf{fix}\,F) \sqsubseteq X & \{\text{def. fix}\} \\ \Leftarrow & \forall X;\ F.X \sqsubseteq X :: F.(\mathsf{fix}\,F) \sqsubseteq F.X & \{\text{assumption } X\} \\ \Leftarrow & \forall X;\ F.X \sqsubseteq X :: \mathsf{fix}\,F \sqsubseteq X & \{F \text{ monotonic}\} \\ \Leftrightarrow & \mathsf{True} & \{\text{def. fix}\} \end{array}$$

3.4 From induction to recursion

Once one has a well-founded relation $\prec$ (or a deterministic rule set), one can define recursive functions provided one has the iota-correspondence of subsection 2.11.1 between functions and single-valued relations. The proof would be somewhat simpler in set theory, as functions and single-valued relations are identified there.

Theorem 3.7 (transfinite recursion) *If $(\prec) : \subseteq T^2$ is well-founded, and one has a recursion step*

$$s(x : T;\ h : U^{|\prec x|}) : U$$

then one can construct a unique $f : U^T$ such that

$$\forall x : T :: fx = s(x; f|_{\prec x}) \ . \tag{3.11}$$

($f|_{\prec x}$ is the restriction of f to $|\prec x|$, which will henceforth be noted just f.)

Proof. Let the infix binary relation $R : \subseteq T \times U$, that is to become the graph of f, be inductively defined as the least relation X such that

$$\forall x : T;\ h : U^{|\prec x|} \ :: \ \forall (y : \prec x :: y\ X\ hy) \Rightarrow x\ X\ s(x; h) \ . \tag{3.12}$$

To prove single-valuedness of R we apply transfinite induction (3.6) to $Px := \exists!\,|x\ R|$ (the predicate stating that x has a unique R-image), and see that $\forall (x : T :: \exists!\,|x\ R|)$ holds provided

$$\forall x : T;\ \forall (z : \prec x :: \exists!\,|z\ R|) :: \exists!\,|x\ R| \ .$$

So assuming $x : T$ and induction hypothesis

$$\forall z : \prec x :: \exists!\,|z\ R| \ , \tag{3.13}$$

we have to prove $\exists!\,|x\ R|$. From (3.13) we get (using iota) a unique

$$g : !(z : \prec x \triangleright |z\ R|) \ . \tag{3.14}$$

Taking then $y := s(x; g)$, one has by (3.14) and R satisfying (3.12) that $x\ R\ y$, so $\exists\,|x\ R|$.

Now supposing some $z{:}\,T$ satisfies $x\ R\ z$ too, we must show $z = y$. By definition of R (3.12), we have $z = s(x; h)$ for some h with $\forall(y{:}\prec x :: y\ R\ hu)$. By uniqueness of g (3.14) it follows that $h = g$, and hence $z = s(x; h) = s(x; g) = y$.

This completes the constructive proof of single-valuedness of R. So let $p(x{:}\,T){:}\,\exists!\,|x\ R|$ be the corresponding proof term, and take $fx := \iota(px)$. Then $fx = s(x; f)$ holds by (3.12).

For uniqueness, assume $gx = s(x; g|_{\prec x})$. Then by transfinite induction one proves $\forall(x{:}\,T :: gx = fx)$. ■

We may note that this theorem can be generalized to dependent types $U{:}\,\mathbf{Type}^T$. We shall do this in theorem 6.2: given a recursion step

$$s(x{:}\,T;\ h{:}\,\Pi(y{:}\prec x :: Uy)){:}\,Ux\ ,$$

there is a unique $f{:}\,\Pi(T; U)$ satisfying (3.11). The proof goes analogous; one replaces $|R|{:}\subseteq T \times U$ by $R{:}\subseteq \Sigma(T; U)$.

3.5 Conclusion

We have seen our language *ADAM* at work in some inductive definitions. In the rest of the thesis, we develop a general theory for inductive types, based on categorical notions which we introduce in the next chapter.