Inductive definitions: automation and application

John Harrison

University of Cambridge Computer Laboratory
New Museums Site
Pembroke Street
Cambridge
CB2 3QG
England
jrh@cl.cam.ac.uk

Abstract. This paper demonstrates the great practical utility of inductive definitions in HOL. We describe a new package we have implemented for automating inductive definitions, based on the Knaster-Tarski fixpoint theorem. As an example, we use it to give a simple proof of the well-founded recursion theorem. We then describe how to generate free recursive types starting just from the Axiom of Infinity. This contrasts with the existing HOL development where several specific free recursive types are developed first.

1 Inductive definitions

Inductive definitions are very common in mathematics, especially in the definition of formal languages used in mathematical logic and programming language semantics. Camilleri and Melham [6] give some illustrative examples. Examples crop up in other parts of mathematics too, e.g. the definition of the Borel hierarchy of subsets of $\mathbb{R}$. A detailed discussion, from an advanced point of view, is given by Aczel [2].

Inductive definitions define a set S by means of a set of *rules* of the form 'if ... then $t \in S$', where the hypothesis of the rule may make assertions about membership in S. These rules are customarily written with a horizontal line separating the hypotheses (if any) from the conclusion. For example, the set of even numbers E might be defined as a subset of the reals by:

$$\frac{}{0 \in E}$$

$$\frac{n \in E}{(n+2) \in E}$$

Read literally, such a definition merely places some constraints on the set E, asserting its 'closure' under the rules, and does not, in general, determine it uniquely. For example, the set of even numbers satisfies the above, but so does the set of natural numbers, the set of integers, the set of rational numbers and even the the whole set of real numbers! But implicit in writing a definition like this is that E is the *least* set which is closed under the rules. It is when understood in this sense that the above defines the even numbers.

This convention, however, needs to be justified by showing that there *is* a least set closed under the rules. A good try is to consider the set of *all* sets which are closed under the rules, and take their intersection. If only we knew that this intersection was closed under the rules, then it would certainly be the least such set. But in general we don't know that, as the following example illustrates:

$$\frac{n \notin E}{n \in E}$$

There are no sets at all closed under this rule! However it turns out that a simple syntactic restriction on the rules is enough to guarantee that the intersection is closed under the rules. Crudely speaking, the hypotheses must make only 'positive' assertions about membership in S. To state this precisely, observe that we can collect together all the rules in a single assertion of the form:

$$\forall x.\, P[S, x] \Rightarrow x \in S$$

The following example for the even numbers should be a suitable paradigm to indicate how:

$$\forall n.\, (n = 0 \vee \exists m.\, n = m + 2 \wedge m \in E) \Rightarrow n \in E$$

If we make the abbreviation $f(S) = \{x \mid P[S, x]\}$ the assertion can be written $f(S) \subseteq S$. Our earlier plan was to take the intersection of all subsets S which have this property, and hope that the intersection too is closed under the rules. A sufficient condition for this is given in the following fixpoint theorem due to Knaster [14] and Tarski [22] (which holds for an arbitrary complete lattice): if $f : \wp(X) \to \wp(X)$ is monotone, i.e. for any $S \subseteq X$ and $T \subseteq X$

$$S \subseteq T \Rightarrow f(S) \subseteq f(T)$$

then if we define

$$F = \bigcap \{S \subseteq X \mid f(S) \subseteq S\}$$

not only is $f(F) \subseteq F$ but F is actually a fixpoint of f, i.e. $f(F) = F$. To see this, define $B = \{S \subseteq X \mid f(S) \subseteq S\}$. Observe that for any $S \in B$ we have $F \subseteq S$, so by monotonicity $f(F) \subseteq f(S)$; but since $f(S) \subseteq S$ we also have $f(F) \subseteq S$. This is true for *any* $S \in B$ so we must also have $f(F) \subseteq F$. Now monotonicity gives $f(f(F)) \subseteq f(F)$, so $f(F) \in B$. This shows that $F \subseteq f(F)$ and putting everything together, $f(F) = F$. Obviously F is the least set closed under the rules, for if F' is another, we have $F' \in B$ and so $F \subseteq F'$.

The fixpoint property $f(F) = F$ yields what we will call a 'cases theorem' because it lists the ways an element of the inductively defined set can arise. In our example, we have:

$$\forall n.\, n \in E \Longleftrightarrow (n = 0 \vee \exists m.\, n = m + 2 \wedge m \in E)$$

This is a bonus from the fact that we actually have a *fixpoint*. It also yields what we will call an 'induction theorem' asserting that F is contained in every set closed under the rules. This justifies proofs by 'structural induction' or 'rule induction' (the terminology is from [25]).

$$\forall E' \subseteq \mathbb{N}.\, (0 \in E' \wedge (\forall n \in \mathbb{N}.\, n \in E' \Rightarrow (n + 2) \in E')) \Rightarrow E \subseteq E'$$

It is easy to derive from the cases theorem by a bit of simple logic that the inductively defined set *is* closed under the rules. We will refer to this as the 'closure theorem'.

$$0 \in E \wedge (\forall n \in \mathbb{N}.\, n \in E \Rightarrow (n + 2) \in E)$$

2 Automating inductive definitions

There are two existing packages for inductive definitions in HOL. One, by Melham [16], which does not use the Knaster-Tarski theorem, is limited to finitary hypotheses, excluding those of the form $(\forall x.\, \ldots) \Rightarrow t \in S$. On the other hand, it provides a very attractive interface and good proof support, and has been extended to support mutual induction by Roxas [21]. The other package was developed by Andersen and Petersen [4] as a byproduct of much more general theories. It is based on Tarski's theorem, and thus more general than Melham's package, but provides no substantial automation — in particular the user must prove the monotonicity theorem manually. Mutual induction has to be dealt with by hand too. More recently, Paulson [18] has produced a package for the Isabelle prover, also based on Knaster-Tarski. Our aim was to combine the best features of the available packages. The implementation can be divided into several stages which we discuss separately:

1. Transition from clausal definition to 'cases' form
2. Proof of monotonicity
3. Application of Tarski's theorem
4. Recovery of 'clausal' form of rules and induction theorem.

It is convenient to broaden the idea of defining a set (or equivalently, unary relation) and allow inductive definitions of arbitrary arities. We shall say that f is monotone iff for each pair of n-ary relations R and R':

$$\begin{aligned}&(\forall x_1, \ldots, x_n.\, R x_1\, \ldots\, x_n \Rightarrow R' x_1\, \ldots\, x_n)\\ &\Rightarrow (\forall x_1, \ldots, x_n.\, f(R) x_1\, \ldots\, x_n \Rightarrow f(R') x_1\, \ldots\, x_n)\end{aligned}$$

The Knaster-Tarski theorem generalizes in the obvious way. Making this generalization has two advantages:

– n-ary relations do not have to be reduced to an uncurried form before they can be defined.
– It fits in well with the automated proofs of monotonicity (see below).

Transition from clausal to casewise form

Each rule is of the form:

$$\forall x_1, \ldots, x_n.\, E[R, x_1, \ldots, x_n] \Rightarrow R\, t_1\ \ldots\ t_k$$

except that the antecedant may be absent (for regularity, the basic package assumes all rules have an antecedant; a wrapper function then inserts $\top$ as an antecedant before calling the main function then modifies the resulting theorems). This gives rise to a casewise version:

$$(\exists x_1, \ldots, x_n.\, (z_1 = t_1) \wedge \ldots \wedge (z_k = t_k) \wedge E[R, x_1, \ldots, x_n]) \Rightarrow R\, z_1\ \ldots\ z_k$$

Each rule $1, \ldots, p$ is transformed into something of this form, i.e.

$$E_i[R, z_1, \ldots, z_k] \Rightarrow R\, z_1\ \ldots\ z_k$$

For the sake of readability of the cases theorem, we avoid introducing local variables when the relevant t_i is itself a variable. However more could be done in other cases, e.g. if t_i is a pair of variables. Now, the resulting composite rule is:

$$E_1[R, z_1, \ldots, z_k] \vee \ldots \vee E_p[R, z_1, \ldots, z_k] \Rightarrow R\, z_1\ \ldots\ z_k$$

Proof of Monotonicity

The proof of monotonicity is managed by a function which makes a recursive traversal over the term. Monotonicity has good compositionality properties over the 'positive' logical operators like conjunction:

$$(A \Rightarrow A') \wedge (B \Rightarrow B') \Rightarrow ((A \wedge B) \Rightarrow (A' \wedge B'))$$

and the universal quantifier (the dollar emphasizes its role as a higher order function):

$$(\forall x.\, P\ x \Rightarrow Q\ x) \Rightarrow (\$\forall\ P \Rightarrow \$\forall\ Q)$$

We have an assignable variable of rules embodying such principles which the function tries to use, so users can add monotonicity rules for special functions they may need. They are stored as rules, not theorems, since some cannot be stored as a single theorem because they take various numbers of arguments (e.g. the rule for `UNCURRY`). However most of the rules just apply `MATCH_MP` to a theorem of the above form. We store some theorems which indicate *antitone* instances, and everything works provided they balance out (e.g. $\neg P \Rightarrow Q$). The monotonicity function works by composing the above rules, using the fact that formulas not involving the relation are trivially monotone, and dealing with abstractions as follows:

$$(\forall x.\, P\ x \Rightarrow Q\ x) \Rightarrow (\forall x\ y.\, (\lambda y.\, P)\ y\ x \Rightarrow (\lambda y.\, Q)\ y\ x)$$

We see here that to get a smooth recursion over the term structure, it is convenient to allow the variadic notion of 'monotonicity'. The function proves monotonicity completely automatically for almost every conceivable instance. For example, the reader may confirm that it deals with paired quantifiers correctly.

Applying Knaster-Tarski

There is little to say about this stage, except that the theorem generalizes to the variadic case in a completely obvious way. The variadic form cannot be stored as a single HOL theorem, but it is simple to implement a derived rule, essentially performing the proof above (alternatively one could store an 'uncurried' form and 'curry' it appropriately each time).

Recovering the clausal forms

The 'cases' theorem pops out of the previous stage in its final form. But for the closure and induction theorems we need to reverse the initial transformation. This is tedious, but perfectly straightforward.

Extension to schemas and mutual induction

Sometimes (e.g. when defining the transitive closure of a relation) there are additional arguments. These can be handled by ignoring them during the main phase of the function and reintroducing them by Skolemization when an existence theorem has been derived. More constructively, the definition as an intersection can be retained explicitly and abstracted over the additional schematic arguments.

Mutually inductive relations can be handled by the basic package using additional 'tag' arguments (e.g. distinct natural numbers) to define a family of relations. However in our implementation we actually generalize the above Tarskian procedure to multiple relations. If we are defining a family of relations $R_1, \ldots, R_k$, then we carry through hypotheses of monotonicity for all the relations together. The rules involving the same relation in their hypothesis are collected together and the definition is modified simply by generalizing over *all* the relations together (i.e. intersecting over the all the relations involved). Essentially the same reasoning as in the one-relation case now goes through.

A foundational remark

All of the above can be carried out in an extremely simple, intuitionistic subsystem of the HOL logic. No extensive logical or theory support is required. This is valuable since, as this paper sets out to show, many theory developments are greatly eased by having a system for inductive definitions available early on.

3 Well-founded relations

A binary relation $\prec$ on a set X is said to be *well-founded* iff every nonempty subset of X has a $\prec$-minimal member, i.e.

$$WF(\prec) = \forall S \subseteq X.\, S \neq \emptyset \Rightarrow \exists m \in S.\, \forall y \in S.\, y \not\prec m$$

Note that despite the suggestive notation and the use of the term 'minimal', we are making no special assumptions about the relation $\prec$. Evidently if $\prec$ is well-founded it must be antisymmetric, (if $x \prec y$ and $y \prec x$ then $\{x, y\}$ would fail to have a minimal member), and so irreflexive (set $x = y$ here). But we do not assume that $\prec$ is transitive, still less connected (it is easy to see that the latter implies the former). This means that $y \not\prec m$ is not, in general, the same as $m \preceq y$.

There are several important consequences of well-foundedness, such as the validity of proofs by induction and definitions by recursion on $\prec$. In fact it is not widely appreciated that all the following are *equivalent*.

1. $\prec$ is well-founded
2. There are no infinite descending ω-chains
3. Well-founded induction holds for $\prec$.
4. Every admissible recursion is satisfied by *no more than one* function.
5. Every admissible recursion is satisfied by *exactly one* function.

An infinite descending ω-chain is a function $s : \mathbb{N} \rightarrow X$ such that $\forall n \in \mathbb{N}.\, s_{n+1} \prec s_n$. The equivalence involving descending ω-chains differs in two respects from the other equivalences. One is that to establish its equivalence involves the Axiom of (Countable Dependent) Choice. To the shamelessly non-constructive HOL user this is of little import; more significant is the fact that it presupposes the existence of the natural numbers, which is something we want to avoid at this stage.

Most of the above are easy to prove, and the proofs are well-known. We want to focus on the recursion theorem, since we can give a simple proof using an inductive definition, and moreover one which makes essential use of the more general hypotheses our system allows. The recursion theorem justifies the existence of a unique function $f : X \rightarrow Y$ such that $\forall x \in X.\, f(x) = \psi(f, x)$, provided each $\psi(f, x)$ depends only on the values of $f(z)$ for $z \prec x$. This condition on a recursion equation is what we called 'admissibility' above. Formally, we can define admissibility of $\psi : (X \rightarrow Y) \times X \rightarrow Y$ with respect to a relation $\prec$ as follows:

$$Adm_{\prec}(\psi) = \forall f\ g.\, (\forall x \in X.\, (\forall z \prec x.\, f(z) = g(z)) \Rightarrow (\psi(f, x) = \psi(g, x))$$

We claimed that

$$\forall \prec.\, WF(\prec) \Rightarrow \forall \psi.\, Adm_{\prec}(\psi) \Rightarrow \exists! f : X \to Y.\, \forall x \in X.\, f(x) = \psi(f, x)$$

To prove this, we make the following inductive definition of a binary relation R on X.

$$\frac{\forall z \prec x.\, (z, g(z)) \in R}{(x, \psi(g, x)) \in R}$$

The cases theorem is:

$$\forall x\ y.\, (x, y) \in R \Longleftrightarrow \exists g.\, (\forall z \prec x.\, (z, g(z)) \in R) \wedge y = \psi(g, x)$$

We claim that R is the graph of the desired recursive function. First we prove by induction on $\prec$ that it is the graph of some function, i.e. $\forall x \in X.\, \exists! y \in Y.\, (x, y) \in R$. Supposing this is true for all $z \prec x$, then we can certainly define a function $g : X \to Y$ so that $\forall z \prec x.\, (z, g(z)) \in R$ (for $z \prec x$ choose for $g(z)$ the unique y with $(z, y) \in R$, and for other z some arbitrary value). Now $(x, \psi(g, x)) \in R$, so we have proved existence. For uniqueness, suppose $(x, y) \in R$ and $(x, y') \in R$. Then by the cases theorem there are functions g and h with:

$$(\forall z \prec x.\, (z, g(z)) \in R) \wedge y = \psi(g, x)$$

and

$$(\forall z \prec x.\, (z, h(z)) \in R) \wedge y' = \psi(h, x)$$

But now by the uniqueness part of the inductive hypothesis, $\forall z \prec x.\, g(z) = h(z)$, and by the admissibility assumption $\psi(g, x) = \psi(h, x)$. Consequently $y = y'$ as required.

Let f be the function with graph R. By the induction rule, we know that $(\forall z \prec x.(z, f(z)) \in R) \Rightarrow (x, \psi(f, x)) \in R$. By definition of f, we know that $\forall x \in X.\, (x, f(x)) \in R$; but this certainly implies the antecedant of this implication. So $\forall x \in X.\ (x, \psi(f, x)) \in R$; but the fact that R is single-valued, already proven, shows that $\forall x \in X.\, f(x) = \psi(f, x)$.

Uniqueness of the function f is a simple induction. More surprising is the converse, that just the *uniqueness* part of the recursion theorem implies that $\prec$ is well-founded. To see this, suppose that

$$\forall \psi.\, Adm_{\prec}(\psi) \Rightarrow \forall f\ g.\, (\forall x.\, f(x) = \psi(f, x)) \wedge (\forall x.\, g(x) = \psi(g, x)) \Rightarrow (f = g)$$

We claim well-founded induction holds. Suppose that for any property P we have $\forall x.\, (\forall z \prec x.\, P(z)) \Rightarrow P(x)$. In the above, set:

$$\begin{aligned}\psi(f,x) &= P(x) \lor \forall z \prec x.\ f(z)\\ f &= P\\ g &= \lambda x.\ \top\end{aligned}$$

in the above theorem. Evidently ψ is admissible, and furthermore $\forall x.(\lambda x.\top)(x) = P(x) \lor \forall z \prec x.(\lambda x.\top)(x)$ is trivially true. But also $\forall x.P(x) = P(x) \lor \forall z \prec x.P(x)$ is trivially true, since it is logically equivalent to $\forall x.\ (\forall z \prec x.\ P(z)) \Rightarrow P(x)$, true by hypothesis. Thus $P = \lambda x.\ \top$ and so $\forall x \in X.\ P(x)$ as required.

These equivalences have all been proved easily in HOL. Actually, the ones involving the recursion theorem can't be stated as single HOL theorems, because they require a local type quantification (over the set Y). To state them as equivalences directly one would need to extend the logic as proposed by Melham [17]. However, as with most other similar troublesome theorems (completeness for first order logic, pointwise continuity in terms of nets etc.) they can easily be split into two parts:

```
|- !L.
    WF L ==>
    (!H.
      (!f g x. (!z. L(z,x) ==> (f z = g z)) ==> (H f x = H g x)) ==>
      (?! f:*->**. !x. f x = H f x))

|- !L.
    (!H.
      (!f g x. (!z. L(z,x) ==> (f z = g z)) ==> (H f x = H g x)) ==>
      (?! f:*->bool. !x. f x = H f x)) ==>
    WF L
```

Note that we are considering relations on the whole type, not a specific subset. However it is clear from the definition that either expanding or contracting the domain of the well-founded relation still yields a well-founded relation.

4 Automating recursive definitions

The well-founded recursion theorem is sometimes quite practically useful. The only derived definitional mechanisms built into the HOL core are for *primitive* recursive functions. Primitive recursion in higher order logic is stronger than in first order logic. For example, the following function ($\psi_1(a,b) = ab$, $\psi_2(a,b) = a^b$ and so on):

$$\begin{aligned}\psi_0(a,b) &= a + b\\ \psi_{n+1}(a,0) &= \begin{cases} 0 \text{ if } n = 0\\ 1 \text{ if } n = 1\\ a \text{ otherwise}\end{cases}\\ \psi_{n+1}(a,b+1) &= \psi_n(a,\psi_{n+1}(a,b))\end{aligned}$$

was introduced by Ackermann [1] [1] precisely to show that some functions which are not first order primitive recursive can nevertheless be defined by higher order primitive recursion (using an iteration functional like ML's `funpow`). Nevertheless for more general recursions like that involved in quicksort, it is attractive to be able to use well-founded recursion in its full generality. The well-founded recursion theorem is an excellent starting point; the fact that we have not assumed $\prec$ to be antisymmetric means it subsumes the use of a separate 'measure' function. However as yet there is no support to define functions without manually instantiating the theorem. Making this process easier has been explored by Ploegaerts et. al. [20] and by van der Voort [23]. A rather different scheme for defining recursive functions in HOL via a theory of CPOs, is described by Agerholm [3]. Recently, Slind has been developing a very useful suite of automated tools for recursive definitions.

5 Free recursive types

Suppose we have any set X, together with an element $z \in X$ and a function $s : X \to X$. We can carve out a subset N by the following inductive definition:

$$\frac{}{z \in N}$$

$$\frac{n \in N}{s(n) \in N}$$

From the inductive definition, we immediately find:

$$z \in N$$

$$\forall n \in N.\, n \in N \Rightarrow s(n) \in N$$

$$\forall P.\, P(z) \wedge (\forall n \in N.\, P(n) \Rightarrow P(s(n))) \Rightarrow \forall n \in N.\, P(n)$$

These are three of the Peano axioms for the natural numbers, with z taking the place of zero and s of successor. We have thus succeeded in inductively defining a set which shares these three important properties of $\mathbb{N}$. However the natural numbers have the additional property that they are *freely* generated by the constructors 0 and SUC, i.e. two values produced from 0 and SUC are the same only if they were constructed in exactly the same way, i.e.

$$\overbrace{SUC(\ldots(SUC(0))\ldots)}^{m\ times} = \overbrace{SUC(\ldots(SUC(0))\ldots)}^{n\ times}$$

if and only if $m = n$. This follows from the other two Peano axioms:

[1] The better-known 2-argument function often called Ackermann's function is actually due to Péter [19].

– The constructors are distinct, i.e. $SUC(n) \neq 0$.
– The constructors are injective, i.e. $(SUC(m) = SUC(n)) \Rightarrow (m = n)$.

Now if $m < n$ in the above equation, repeatedly using injectivity yields:

$$\overbrace{SUC(\ldots(SUC(0))\ldots)}^{n-m\ times} = 0$$

which contradicts distinctness. If we knew that z and s had these two properties, the set N would obey all the (second order) Peano axioms and hence be isomorphic to $\mathbb{N}$. The detailed nature of x, s and X are irrelevant. In fact the Axiom of Infinity in HOL uses the Dedekind definition of an infinite set: namely there exists a function from the set to itself which is injective but not surjective. We can take this function (eventually restricted to $\mathbb{N}$) as SUC, and any point not in its range as 0. (This is how the type `:num` is currently developed in HOL, though it does not explicitly use an inductive definition.)

From the Peano axioms, a theorem justifying primitive recursion may be derived[2]. For any set X, element $a \in X$ and function $\psi : X \times \mathbb{N} \to X$, there is a unique function $f : \mathbb{N} \to X$ such that $f(0) = a$ and $\forall n \in \mathbb{N}.\ f(SUC(n)) = \psi(f(n), n)$. Once again the easiest proof by far is to build up the graph of the function by an inductive definition:

$$\frac{}{(0, a) \in f}$$

$$\frac{(n, y) \in f}{(SUC(n), \psi(y, n)) \in f}$$

The details are much as in the well-founded case and are left to the reader. And just as in the well-founded case, the recursion theorem implies all the Peano axioms (the fact that $0 \in \mathbb{N}$ and $n \in \mathbb{N} \Rightarrow SUC(n) \in \mathbb{N}$ are implicit in the types of the functions concerned). The uniqueness part proves that induction holds, and now the existence part proves freeness. For example, the recursion theorem allows a function $Z : \mathbb{N} \to$ *bool* to be defined with $Z(0) = \top$ and $Z(SUC(n)) = \bot$, showing that the constructors are distinct. Likewise we can define a 'predecessor' function and so show injectivity.

[2] Often called an 'initiality' theorem since it asserts that $\mathbb{N}$ is an initial object in the category of $(0,SUC)$-algebras (or equivalently a free SUC-algebra on one generator), i.e. there is a unique homomorphism from $\mathbb{N}$ into any other $(0,SUC)$-algebra. Properly speaking, a statement of initiality does not involve the extra argument n to ψ, but since we are talking about the category of *all* algebras with the same signature, the two are equivalent: set $O_A = (0, a)$ and $SUC_A(n, y) = (SUC(n), \psi(y, n))$. For more on these matters see [8].

6 Automating recursive types

Given the inductive definitions package, we are in a strong position to automate recursive types as described above. We use inductive definitions first to carve out a subset of an existing type to yield the new one, and also to justify the recursion theorem. However not all inductive definitions are as simple as the one for natural numbers. There may be many constructors, which may be mutually recursive and even nested with each other and previously defined constructors. They are perhaps best illustrated by the following example, which is meant to be a type of terms for an embedded syntax of first order logic, with the variables parametrized by natural numbers and the function symbols by integers.

```
Term = Var num | Fun int (Term list)
```

Now we can avoid having to deal with existing constructors like `list` explicitly by defining an isomorphic type by mutual recursion and then modifying the theorems at the end, as pointed out by Gunter [9]. Furthermore, nested instances of constructors can be unwound by introducing locally some additional types.

```
Term = Var num | Fun int Termlist
Termlist = Nil | Cons Term Termlist
```

First we collect together all the basic types $\alpha_1, \ldots, \alpha_n$ (here num and int). All we need to generate some distinct injective constructors is a set u with injective functions:

- $i_\alpha : \alpha \to u$ to inject the basic types.
- $t : \mathit{ind} \to u$ to inject distinct tag elements to distinguish the constructors, of which there may be arbitrarily many.
- $P : u \to u \to u$ to implement multivariate constructors by iteratively pairing up the arguments.

This is pretty easy to arrange, by perfectly straightforward cardinality reasoning. Without any nontrivial use of arithmetical properties, we can show that $X = \mathit{ind} \to \mathit{bool}$ admits an injective function $\mathit{bool} \to X \to X$; thus $Y = X \to \mathit{bool}$ admits one $Y \to Y \to Y$. Now we may take $Y \to \alpha_1 \to \cdots \to \alpha_n \to \mathit{bool}$ as the set u. For our example, we can define the constructors on u as follows:

$$\begin{aligned}
Var\ n &= P\ (t\ 0)\ (i_{num}(n)) \\
Fun\ i\ l &= P\ (t\ 1)\ (P\ (i_{int}(i))\ l) \\
Nil &= P\ (t\ 2)\ arb \\
Cons\ x\ l &= P\ (t\ 3)\ (P\ x\ l)
\end{aligned}$$

Clearly these are distinct and injective, because P and all the i_X are. We may now define sets to represent $Term$ and $Termlist$ by mutual induction. The new types may then be declared (since $Termlist$ and $(Term)list$ are isomorphic, we can freely transform the former into the latter with a little effort). The recursion theorem may then be proved, again using an inductive definition.

And now, if we want to consider an extension to infinitely branching trees as Gunter [10] does, finding a corresponding universe is almost as easy. It suffices to create a new universe v with an injection $(u \to v) \to v$. It's easy to see that $u \to u$ suffices: given a function $f : u \to (u \to u)$, form the function $(UNCURRY\ f) \circ P^{-1}$. In cardinal terms, $|u \to v| = |u \to (u \to u)| = |u \to u|^{|u|} = (|u|^{|u|})^{|u|} = |u|^{|u|^2} = |u|^{|u|} = |u \to u| = |v|$, using the fact that we already know $|u|^2 = |u|$. This contrasts with the explicit construction in Gunter's work. On the other hand, it may be that to store a form of the recursion theorem which allows explicit instantiation, it may be useful to carve out an inductive subset of the constructed space right from the beginning.

Conclusions

We have developed a powerful and easy-to-use package for automating inductive definitions, and demonstrated the power of such definitions in important theory developments. As yet the proposed recursive types package has not been implemented, but we believe the idea holds considerable promise. The merit of our approach is that finding large enough underlying sets is done very generally using abstract set theory, so we avoid creating specific recursive types in a piecemeal way. It is intended to experiment with this more 'rational' theory development in the `gtt` system [11]. It also makes easier certain other changes, e.g. introducing a binary system for numeral constants along the lines of the existing numeral library [15].

Acknowledgements

I am grateful to my supervisor, Mike Gordon, and to the EPSRC and the Newton Trust for financial support. Tom Melham's original package provided the inspiration, and I also learned a lot about free recursive types from Tom's work and conversation. (The proof that well-founded recursion implies well-foundedness was inspired by Tom's analogous HOL proofs for free recursive types.) The idea of creating recursive types using an inductive definition came from Larry Paulson's work. The possibility of storing a general form of the recursion theorem is due to Vernon Austel. The idea of using inductive definitions to prove recursion theorems came from the proof of the recursion theorem for $\mathbb{N}$ in [12]. My interest in well-founded relations in general was sparked by Konrad Slind. I have also benefited greatly from conversations with Elsa Gunter. Malcolm Newey impressed on me the aesthetic appeal of leaving numbers till late in the theory development. Comments from the anonymous referees were very helpful.

References

1. Wilhelm Ackermann. Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematische Annalen*, 99:118–133, 1928. English translation, 'On Hilbert's construction of the real numbers', in [24], pp. 493–507.
2. Peter Aczel. An introduction to inductive definitions. In J. Barwise and H.J. Keisler, editors, *Handbook of mathematical logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 739–782. North-Holland, 1991.
3. Sten Agerholm. A HOL basis for reasoning about functional programs. BRICS Report Series RS-94-44, Centre of the Danish National Research Foundation, Department of Computer Science, University of Aarhus, Ny Munkegade, DK-8000 Aarhus C, Denmark, 1994.
4. Flemming Andersen and Kim Dam Petersen. Recursive boolean functions in HOL. In Archer et al. [5], pages 367–377.
5. Myla Archer, Jeffrey J. Joyce, Karl N. Levitt, and Phillip J. Windley, editors. *Proceedings of the 1991 International Workshop on the HOL theorem proving system and its Applications*, University of California at Davis, Davis CA, USA, 1991. IEEE Computer Society Press.
6. Juanito Camilleri and Tom Melham. Reasoning with inductively defined relations in the HOL theorem prover. Technical Report 265, University of Cambridge Computer Laboratory, New Museums Site, Pembroke Street, Cambridge, CB2 3QG, UK, 1992.
7. Luc J. M. Claesen and Michael J. C. Gordon, editors. *Proceedings of the IFIP TC10/WG10.2 International Workshop on Higher Order Logic Theorem Proving and its Applications*, volume A-20 of *IFIP Transactions A: Computer Science and Technology*, IMEC, Leuven, Belgium, 1992. North-Holland.
8. J. A. Goguen, J. W. Thatcher, and E. G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In Raymond T. Yeh, editor, *Current Trends in Programming Methodologies, volume IV*, pages 80–149. Prentice-Hall, 1978.
9. Elsa L. Gunter. Why we can't have SML style datatype declarations in HOL. In Claesen and Gordon [7], pages 561–568.
10. Elsa L. Gunter. A broader class of trees for recursive type definitions for HOL. In Joyce and Seger [13], pages 141–154.
11. John Harrison and Konrad Slind. A reference version of HOL. Presented in poster session of 1994 HOL Users Meeting and only published in participants' supplementary proceedings. Available on the Web from `http://www.dcs.glasgow.ac.uk/~hug94/sproc.html`, 1994.
12. Nathan Jacobson. *Basic Algebra I*. W. H. Freeman, 2nd edition, 1989.
13. Jeffrey J. Joyce and Carl Seger, editors. *Proceedings of the 1993 International Workshop on the HOL theorem proving system and its applications*, volume 780 of *Lecture Notes in Computer Science*, UBC, Vancouver, Canada, 1993. Springer-Verlag.
14. B. Knaster. Un théorème sur les fonctions d'ensembles. *Annales de la Société Polonaise de Mathématique*, 6:133–134, 1927. Volume published in 1928.
15. Tim Leonard. The HOL numeral library. Distributed with HOL system, 1993.
16. Thomas F. Melham. A package for inductive relation definitions in HOL. In Archer et al. [5], pages 350–357.
17. Thomas F. Melham. The HOL logic extended with quantification over type variables. In Claesen and Gordon [7], pages 3–18.

18. Lawrence C. Paulson. A fixedpoint approach to implementing (co)inductive definitions. Technical Report 320, University of Cambridge Computer Laboratory, New Museums Site, Pembroke Street, Cambridge, CB2 3QG, UK, 1993.
19. R. Péter. Konstruktion nichtrekursiver Funktionen. *Mathematische Annalen*, 111:42–60, 1935.
20. Wim Ploegaerts, Luc Claesen, and Hugo De Man. Defining recursive functions in HOL. In Archer et al. [5], pages 358–366.
21. Rachel E. O. Roxas. A HOL package for reasoning about relations defined by mutual induction. In Joyce and Seger [13], pages 129–140.
22. Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
23. Mark van der Voort. Introducing well-founded function definitions in HOL. In Claesen and Gordon [7], pages 117–132.
24. Jean van Heijenoort, editor. *From Frege to Gödel: A Source Book in Mathematical Logic 1879–1931.* Harvard University Press, 1967.
25. Glynn Winskel. *The formal semantics of programming languages: an introduction.* Foundations of computing. MIT Press, 1993.

This article was processed using the LaTeX macro package with LLNCS style