Computer Science 15-312
Inductive Definitions

Robert Harper

Spring, 2000

1 Introduction

It is common in the study of programming languages to define a set by a collection of *rules* that define the members of that set. Each rule has zero or more *premises*, or *requirements*, and one *conclusion*.

Example 1 (Numerals, first attempt)
We may define the set of unary numerals (i.e., numerals in base 1) for the natural numbers as follows:

1. **Zero** *is a numeral;*

2. *if n is a numeral, then* $\mathbf{Succ}(n)$ *is also a numeral.*

Equivalently, we might say that the set $\mathbf{N}$ *of numerals is defined by the following rules:*

1. $\mathbf{Zero} \in \mathbf{N}$*;*

2. *if* $n \in \mathbf{N}$*, then* $\mathbf{Succ}(n) \in \mathbf{N}$.

Observe that in each formulation the first rule has no premises, whereas the second has one.

Example 2 (Binary Trees, first attempt)
We may define the set of binary trees by the following rules:

1. *The empty tree,* **Empty***, is a binary tree.*

2. *If t_l and t_r are binary trees, then so is* $\mathbf{Node}(t_l, t_r)$.

Equivalently, we might say that the set $\mathbf{T}$ *of binary trees is defined by the following rules:*

1. $\mathbf{Empty} \in \mathbf{T}$*;*

2. *if* $t_l \in \mathbf{T}$ *and* $t_r \in \mathbf{T}$*, then* $\mathbf{Node}(t_1, t_2) \in \mathbf{T}$*.*

The first rule has no premises; the second has two.

When thinking about these rules you might well ask yourself what exactly is a node, or what are zero and successor? These are examples of *value constructors*, the "atoms" out of which we build "molecular" data structures. Technically we should say precisely what we mean by these constants and constructors, but for now we'll just assume that there is some way to define them, say in terms of sets.

In ML value constructors are a primitive concept. They are introduced by *datatype* declarations such as these, which closely parallel the definitions given above.

```
datatype num = Zero | Succ of num
datatype tree = Empty | Node of tree * tree
```

These declarations introduce the types `num` and `tree`, respectively, in terms of the given value constructors. Each constructor implicitly defines a "rule" for membership in that type. The constructors `Zero` and `Empty` have no arguments; this corresponds to a rule with no premises. The constructor `Succ` has one argument, corresponding to the single premise in the inductive definition; the constructor `Node` has two arguments, corresponding to the two premises in the inductive definition.

Exercise 3
Give a collection of rules defining the set of strings over an alphabet Σ*. Give a corresponding ML* `datatype` *declaration for the same set.*

When we say that a set is defined by a set of rules, what precisely do we mean? Which set do we consider to be defined by those rules? To see why this is an important question, consider the set

$$\mathbf{N}^* = \{\,\mathbf{Zero}, \mathbf{Succ}(\mathbf{Zero}), \mathbf{Succ}(\mathbf{Succ}(\mathbf{Zero})), \ldots\,\} \cup \{\,\infty\,\},$$

where $\infty = \mathbf{Succ}(\mathbf{Succ}(\ldots))$ is an "infinite" stack of successors, which satisfies the equation $\mathbf{Succ}(\infty) = \infty$. Observe that $\mathbf{Zero} \in \mathbf{N}^*$ and that if $n \in \mathbf{N}^*$, then $\mathbf{Succ}(n) \in \mathbf{N}^*$ — that is, $\mathbf{N}^*$ meets the requirements that "define" **N**. Yet clearly $\mathbf{N}^* \supsetneq \mathbf{N}$, and hence these requirements do not, by themselves, pick out a unique set.

Incidentally, if you're skeptical about whethere there exists an element ∞ described above, here's a way to define it in ML, using a "lazy" `datatype` declaration:[1]

```
datatype lazy num_star = Zero | Succ of num_star
```

Here's ∞, spelled out in ML code:

```
val rec lazy infty = Succ infty
```

Here's the predecessor function:

```
fun pred (Succ n) = n
```

You can check by experimetation that `infty` really is an infinite stack of successors as follows:

```
val Succ infty' = infty
val Succ infty'' = infty'
val Succ infty''' = infty''
```

and so on. Each `val` binding strips off one level of `Succ`, leaving another "copy" of infinity behind — $\infty - 1 = \infty$!

For a collection of rules rules to define a set, we must say something more than the set must obey these rules. We need an *extremal clause* that states that "nothing else is in the set except those elements that are required to be there by the rules." This may sound like a bit of legalese, but mathematically it is essential to include the extremal clause in order to rule out unintended interpretations of the rules such as we say with the set $\mathbf{N}^*$. Thus, the definition of **N** should really be stated as follows:

Example 4 (Numerals, revised)
*The set **N** is defined by the following rules:*

1. **Zero** *is a numeral;*

2. *if n is a numeral, then* $\mathbf{Succ}(n)$ *is also a numeral;*

3. *nothing else is a numeral.*

[1] This is valid only for SML/NJ version 110.9 or later. To enable the lazy evaluation primitives, enter `Compiler.Control.lazysml := true` at top level.

Equivalently, we say that $\mathbf{N}$ *is the* least, *or* smallest, *set defined by the first two rules, by which we mean precisely that nothing else is in the set except as is forced by the rules.*

To see that $\mathbf{N}^*$ is ruled out by the extremal clause, observe that ∞ has no business being in the specified set because it is not forced to be in there by the rules. More precisely, since $\mathbf{N} \subsetneq \mathbf{N}^*$, it follows that $\mathbf{N}^*$ is not the least set closed under the rules for **Zero** and **Succ**(), and hence $\mathbf{N}^*$ is ruled out as a possible interpretation of the preceding definition.

Similarly, we may revise the definition of binary trees by adding an extremal clause as follows.

Example 5 (Binary Trees, revised)
We may define the set of binary trees by the following rules:

1. *The empty tree,* **Empty**, *is a binary tree.*

2. *If* t_l *and* t_r *are binary trees, then so is* $\mathbf{Node}(t_l, t_r)$.

Equivalently, we might say that the set $\mathbf{T}$ *of binary trees is defined by the following rules:*

1. $\mathbf{Empty} \in \mathbf{T}$;

2. *if* $t_l \in \mathbf{T}$ *and* $t_r \in \mathbf{T}$, *then* $\mathbf{Node}(t_1, t_2) \in \mathbf{T}$;

3. *nothing else is a binary tree.*

The first rule has no premises; the second has two.

In the presence of the extremal clause, a collection of rules of the kind given above determines a unique set. In practice we do not explicitly state the extremal clause, but rather we state that the set in question is *inductively defined* by a given collection of rules. For example, we may say that the set $\mathbf{N}$ is inductively defined by the two rules membership rules given at the beginning of these notes. In doing so we are implicitly stating that nothing else is to be a member of that set unless it is "forced" to be there by the rules.

Since the smallest set inductively defined by a set of rules contains only those elements explicitly required by the rules, we may envision elements of this set as being built up in "stages". At the first stage we introduce elements that are forced to be in the set by some rule with no premises. (For example, **Empty** would appear at the first stage of the construction of the set $\mathbf{T}$.) Suppose that we have completed stage s of the construction. At stage $s+1$ we add in those elements that are introduced by a rule whose premises are satisfied at stage s. (For example, if t_l and t_r are constructed at stage s, then $\mathbf{Node}(t_l, t_r)$ is constructed at stage $s+1$.) We may then think of the set inductively defined by these rules as the set of elements that appear at some stage s in the construction.

2 Rule Induction and Recursion

Why do we say that the least set defined by a collection of rules is "inductively" defined by it? As the terminology suggests, the answer is that there is a close connection with reasoning by mathematical induction. Here's why. Suppose that we wish to prove that every binary tree t has a *height* h satisfying the following two requirements:

1. the height of **Empty** is 0;
2. if t_l and t_r have heights h_l and h_r, respectively, then the height of $\mathbf{Node}(t_l, t_r)$ is $1 + \max(h_l, h_r)$.

We call these two conditions the *specification* of the height of a binary tree.

The question is this: how do we know that every binary tree in fact has a height? This might seem like an odd question at first, but consider that the "infinite" binary tree

$$\mathbf{Node}(\mathbf{Node}(\ldots,\ldots), \mathbf{Node}(\ldots,\ldots))$$

has no height in the sense specified! Luckily, the extremal clause rules out such "infinite trees", which makes it possible to assign a height to each binary tree.

How do we prove that every binary tree has a height? That is, how do we prove that for every binary tree $t \in \mathbf{T}$, there exists a natural number h satisfying the specification of the height of a binary tree. Since $\mathbf{T}$ is inductively defined, we proceed by analyzing the rules that define it. By the first rule defining binary trees, t might be **Empty**. But t clearly has a height, namely 0, as given in the specification. By the second rule defining binary trees, t might have the form $\mathbf{Node}(t_l, t_r)$, where t_l and t_r are also binary trees. By induction we may assume that they each have a height, say h_l and h_r, respectively. But then the height h of t is uniquely determined by the equation $h = 1 + \max(h_l, h_r)$, as required by the specification. Since $\mathbf{T}$ contains no elements other than these, we may conclude that every binary tree $t \in \mathbf{T}$ has a height h.

The principle of proof we have just used is called *proof by rule induction*. The proof proceeds by analyzing the possible forms of t (as given by the rules defining the set of binary trees), and proving that in each case t has a well-defined height. In doing so we may *assume* (for the sake of induction) that the "sub-trees" occurring within t have well-defined heights, which we then use to assign a height to t.

All this may seem rather obvious to you. What is important is that you recognize that it is necessary to prove that every tree has a height. Here's another perspective on the same issue, this time drawn from your programming experience. Here's the `height` function written in ML:

```
fun height Empty = 0
  | height (Node (tl, tr)) =
    1 + max (height tl, height tr)
```

Why does this define a function from trees to integers? Given a tree t, is there a unique integer n such that `height`t evaluates to n? This may be proved by induction on the rules defining trees. If t = `Empty`, then `height(`t`)` evaluates to 0. If, on the other hand, t = `Node(`t_l`,`t_r`)`, then we may assume for the induction that `height(`t_l`)` evaluates to h_l and that `height(`t_r`)` evaluates to h_r. It follows that `height(`t`)` evaluates to $1 + \max(h_l, h_r)$, which is uniquely determined from h_l and h_r. This completes the proof.

It is important to notice the close interplay between recursive function definitions and inductive proofs illustrated here. Proving by induction and programming by recursion are two sides of the same coin. The concept of an *invariant specification* for a recursive function (describing the input/output behavior of that function) corresponds exactly to the idea of an *inductive hypothesis* for an inductive proof. Cases in a recursive function that do not make recursive calls correspond to the base cases of inductive proofs. Cases that do make recursive calls correspond to inductive steps.

Exercise 6
Using the lazy datatype primitives of SML, define a type `tree_star` *of (possibly infinite) binary trees, and exhibit an infinite binary tree,* `inf_tree`*, of type* `tree_star` *(there are many possible solutions). Define an analogue of the* `height` *function for the type* `tree_star`*, and observe that* `height inf_tree` *does not terminate — the infinite tree has no height.*

Recall that elements of an inductively-defined set may be considered to be constructed in stages. The elements of the recursive type are those that arise at some finite stage of construction. Using this, we may replace a proof by induction on the rules generating an object o by a proof by ordinary mathematical induction on the stage s at which o is constructed. For example, we may prove that every tree has a height by induction on the stage at which the tree is introduced. At stage 0 we have only the empty tree, whose height is 0. At stage $s+1$, we have in addition those nodes whose children are trees at stage s. By the inductive hypothesis the children of a node have a height, so the node must also have a height. The proof is complete because we have considered all possible stages s. As you can see considering stages is really just a roundabout way of reasoning directly by rule induction. However, it is occasionally helpful to keep the stage interpretation in mind.

3 Mathematical Details [Optional]

For the mathematically-inclined student we will give a set-theoretic account of inductive definitions of subsets of some universe $\mathcal{U}$. The general theory of inductive definitions does not rely on any particular properties of $\mathcal{U}$, but for the sake of examples we shall need to assume that $\mathcal{U}$ is rich enough to include objects of interest and over which suitable value constructors may be defined. The existence of a suitable universe for our purposes is not entirely obvious. Suffice it to say that we may take $\mathcal{U}$ to be the set $\mathbb{N}$ of natural numbers. Using techniques introduced by Gödel, we may encode finitary objects such as trees as natural numbers. This method of assigning unique numbers to each data structure is called "Gödel numbering" in honor of Gödel's discovery.

An inductive definition consists of a set of rules. A *rule* is a pair (X, x), where X is a finite[2] subset of $\mathcal{U}$ and $x \in \mathcal{U}$. The idea is that X is the set of requirements, or premises, of the rule, and x is its conclusion. Note that X might be the empty set, in which case the rule has no premises. A rule of the form $(\{\, x_1, \ldots, x_n \,\}, x)$ is usually written $x_1, \ldots, x_n \to x$ or as the figure

$$\frac{x_1 \quad \ldots \quad x_n}{x}.$$

(When $n = 0$ we drop the arrow or horizontal line entirely.) A *rule set*, $\mathcal{R}$, is a (finite or infinite) set of rules.

[2]This restriction is not strictly necessary, but is sufficient for our purposes. One can also consider transfinite inductive definitions, with no significant changes to the development.

Example 7
Here is a rule set corresponding to the inductive definition of numerals:

$$\begin{aligned}
\mathcal{R}_{\mathbf{N}} &= \{\,(\emptyset, \mathbf{Zero})\,\} \cup \{\,(\{\,n\,\}, \mathbf{Succ}(n)) \mid n \in \mathcal{U}\,\} \\
&= \{\,\mathbf{Zero}\,\} \cup \{\,n \to \mathbf{Succ}(n) \mid n \in \mathcal{U}\,\} \\
&= \{\,\overline{\mathbf{Zero}}\,\} \cup \{\,\frac{n}{\mathbf{Succ}(n)}\,\}
\end{aligned}$$

Notice that this set consists of an infinite set of rules!

Exercise 8
Give a rule set corresponding to the inductive definitions of binary trees and of strings.

A subset $A \subseteq \mathcal{U}$ is *closed* under rules $\mathcal{R}$ (or, is $\mathcal{R}$*-closed*) iff for each rule $X \to x$ in $\mathcal{R}$, if $X \subseteq A$, then $x \in A$. That is, A contains all of the elements introduced by an application of a rule in $\mathcal{R}$; it might also contain more than just these.

Exercise 9
Check that both $\mathbf{N}$ *and* $\mathbf{N}^*$ *are closed under* $\mathcal{R}_{\mathbf{N}}$.

We associate with each rule set $\mathcal{R}$ a monotone operator $\Phi_{\mathcal{R}} : Pow(\mathcal{U}) \to Pow(\mathcal{U})$ defined as follows:

$$\Phi_{\mathcal{R}}(A) = \{\,x \mid X \to x \in \mathcal{R},\ X \subseteq A\,\}$$

The operator $\Phi_{\mathcal{R}}$ corresponds to "one turn of the crank" — the set of elements that arise by applying some rule in $\mathcal{R}$ to the elements of the set A. Observe that a set $A \subseteq \mathcal{U}$ is closed under $\mathcal{R}$ iff $\Phi_{\mathcal{R}}(A) \subseteq A$. That is, A contains all of the elements introduced by a single application of a rule in $\mathcal{R}$. For example, $\Phi_{\mathcal{R}_{\mathbf{N}}}(A) = \{\,\mathbf{Zero}\,\} \cup \{\,a \mid a \in A\,\}$.

It is easy to check that $\Phi_{\mathcal{R}}$ is *monotone*: if $A \subseteq B$, then $\Phi_{\mathcal{R}}(A) \subseteq \Phi_{\mathcal{R}}(B)$. Intuitively, if $A \subseteq B$, then the elements emitted by the rules in $\mathcal{R}$ starting B must include at least those emitted by starting from A.

The set *inductively defined* by a rule set $\mathcal{R}$, written $\mathcal{I}(\mathcal{R})$, is defined as follows:

$$\mathcal{I}(\mathcal{R}) = \bigcap \{\,A \subseteq \mathcal{U} \mid \Phi_{\mathcal{R}}(A) \subseteq A\,\}$$

That is, $\mathcal{I}(\mathcal{R})$ is the intersection of all $\mathcal{R}$-closed sets A.

Theorem 10
Let $\mathcal{R}$ *be a rule set and let* $I = \mathcal{I}(\mathcal{R})$.

1. *If A is $\mathcal{R}$-closed, then $I \subseteq A$.*

2. *I is $\mathcal{R}$-closed.*

That is, I is the least set closed under the rules $\mathcal{R}$.

Proof: Let $\Phi = \Phi_{\mathcal{R}}$. If $\Phi(A) \subseteq A$, then $I \subseteq A$ since I is the intersection of all such sets A. This establishes (1). To show that $\Phi(I) \subseteq I$, it is sufficient to show that $\Phi(I) \subseteq A$ whenever $\Phi(A) \subseteq A$, since I is the intersection of all such A's. If $\Phi(A) \subseteq A$, then by (1), $I \subseteq A$. Hence by monotonicity, $\Phi(I) \subseteq \Phi(A)$. But then $\Phi(I) \subseteq A$, as required. This establishes (2). ■

Exercise 11

Prove that $I \subseteq \Phi(I)$, and hence that $I = \Phi(I)$. Hint: *Show that $\Phi(I)$ is $\mathcal{R}$-closed, using part (2) of the Theorem.*

OK, so we've now figured out precisely which set is inductively defined by a collection of rules — it's the intersection of all sets closed under these rules. What matters about this set is what's stated in the Theorem, namely that it's the smallest set closed under the given rules. Why is this important? Because this is how we justify the proofs by rule induction!

Let I be the set inductively defined by some collection of rules $\mathcal{R}$. Suppose we wish to prove that some property holds of every element $x \in I$. Thinking of the property as a subset $P \subseteq I$ (namely, the set of those elements of I that have that property), then what we're trying to do is to prove that $I \subseteq P$ — that is, that everything in I has the property P. The theorem tells us that it is sufficient to show that P is closed under R, since I is the smallest such set. That is, we must show that $\Phi(P) \subseteq P$.

Let's see what this means by considering an example. Let $\mathbf{T}$ be the set of binary trees inductively defined by the rules in the previous section. Let P be the following set

$$P = \{\, t \in \mathbf{T} \mid \textit{there exists } h \textit{ such that } t \textit{ has height } h \,\}$$

where *t has height h* iff either $t = \mathbf{Empty}$ and $h = 0$ or $t = \mathbf{Node}(t_l, t_r)$, t_l has height h_l, t_r has height h_r, and $h = 1 + \max(h_l, h_r)$. To show that every binary tree has a height, it suffices to show that P is closed under the rules defining the set of binary trees. That is, we must show

1. there exists h such that **Empty** has height h (namely, 0);

2. if there exists h_l such that t_l has height h_l and there exists h_r such that t_r has height h_r, then there exists h such that t has height h (namely, $h = 1 + \max(h_l, h_r)$).

This is precisely the argument we gave in the previous section!

Exercise 12
Show that every numeral stands for a unique natural number such that **Zero** *stands for* 0*, and if* n *stands for* k*, then* $\mathbf{Succ}(n)$ *stands for* $k+1$*. Spell out the property* P *in question, and spell out the conditions for* P *to be closed under the rules defining the numerals. Then prove that* P *is closed under these rules.*

To finish up let's make precise the description of $\mathcal{I}(\mathcal{R})$ as being built up in stages. Fix a rule set $\mathcal{R}$, and let $\Phi = \Phi_{\mathcal{R}}$ and $I = \mathcal{I}(\mathcal{R})$. Define the sets I_s for $s \geq 0$ as follows:

$$\begin{array}{rcl} I_0 & = & \emptyset \\ I_{s+1} & = & \Phi(I_s) \end{array}$$

At each stage we "turn the crank" once to add in all elements required by rules applied to the elements at the previous stage. Observe that $I_s \subseteq I_{s+1}$, for every $s \geq 0$. Now define

$$I_\infty = \bigcup_{s \geq 0} I_s$$

Thus the elements of I_∞ are those that arise at some finite stage.

Exercise 13

1. *Prove by induction on* $s \geq 0$ *that* $I_s \subseteq I$*. Conclude that* $I_\infty \subseteq I$*.*

2. *Prove that* $I \subseteq I_\infty$ *by showing that* I_∞ *is* Φ*-closed. Hint: If* $x \in \Phi(I_\infty)$*, then there exists some rule* $X \to x$ *such that* $X \subseteq I_\infty$*. But since* X *is finite (by definition of a rule), there must exist a finite stage* $s \geq 0$ *such that* $X \subseteq I_s$*. Hence* $x \in \Phi(I_s) = I_{s+1}$*. Then apply the previous result.*