Lecture Notes on Inductive Definitions

15-312: Foundations of Programming Languages
Frank Pfenning

Lecture 2
August 28, 2003

These supplementary notes review the notion of an inductive definition and give some examples of rule induction. References to Robert Harper's draft book on *Programming Languages: Theory and Practice* are given in square brackets, by chapter or section.

Given our general goal to define and reason about programming languages, we will have to deal with a variety of description tasks. The first is to describe the grammar of a language. The second is to describe its static semantics, usually via some typing rules. The third is to describe its dynamic semantics, often via transitions of an abstract machine. On the surface, these appear like very different formalisms (grammars, typing rules, abstract machines) but it turns out that they can all be viewed as special cases of *inductive definitions* [Ch. 1]. Following standard practice, inductive definitions will be presented via judgments and inference rules providing evidence for judgments.

The first observation is that context-free grammars can be rewritten in the form of inference rules [Ch. 4.1]. The basic judgment has the form

$$s\ A$$

where s is a string and A is a non-terminal. This should be read as the judgment that *s is a string of syntactic category A*.

As a simple example we consider the language of properly matched parentheses over the alphabet $\Sigma = \{\,\texttt{(}, \texttt{)}\,\}$. This language can be defined by the grammar

$$M ::= \varepsilon \mid M\,M \mid \texttt{(}M\texttt{)}$$

with the only non-terminal M. Recall that ε stands for the empty string.

Rewritten as inference rules we have:

$$\frac{}{\varepsilon\ M} \tag{1}$$

$$\frac{s_1\ M \qquad s_2\ M}{s_1\, s_2\ M} \tag{2}$$

$$\frac{s\ M}{\texttt{(}s\texttt{)}\ M} \tag{3}$$

Our interpretation of these inference rules as an inductive definition of the judgment $s\ M$ for a string s means:

> *$s\ M$ holds if and only if there is a deduction of $s\ M$ using rules (1), (2), and (3).*

Based on this interpretation we can prove properties of strings in the syntactic category M by rule induction. Here is a very simple example.

Theorem 1 (Counting Parentheses)
If $s\ M$ then s has the same number of left and right parentheses.

Proof: By rule induction. We consider each case in turn.

(Rule 1) Then $s = \varepsilon$.

s has 0 left and 0 right parens	Since $s = \varepsilon$

(Rule 2) Then $s = s_1\, s_2$.

$s_1\ M$	Subderivation
$s_2\ M$	Subderivation
s_1 has n_1 left and right parens for some n_1	By i.h.
s_2 has n_2 left and right parens for some n_2	By i.h.
s has $n_1 + n_2$ left and right parens	Since $s = s_1\, s_2$

(Rule 3) Then $s = \texttt{(}s'\texttt{)}$.

$s'\ M$	Subderivation
s' has n' left and right parens for some n'	By i.h.
s has $n'+1$ left and right parens	Since $s = \texttt{(}s'\texttt{)}$

■

The grammar we gave, unfortunately, is ambiguous [Ch. 4.2]. For example, there are infinitely many derivations that $\varepsilon\ M$, because

$$\varepsilon = \varepsilon\,\varepsilon = \varepsilon\,\varepsilon\,\varepsilon = \cdots$$

In the particular example of this grammar we would be able to avoid rewriting it if we can show that the abstract syntax tree [Ch. 5.1] we construct will be the same, independently of the derivation of a particular judgment.

An alternative is to rewrite the grammar so that it defines the same language of strings, but the derivation of any particular string is uniquely determined. In order to illustrate the concept of simultaneous inductive definition, we use two non-terminals L and N, where the category L corresponds to M, while N is an auxiliary non-terminal.

$$\begin{array}{rcl} L & ::= & \varepsilon \mid N\,L \\ N & ::= & \texttt{(}L\texttt{)} \end{array}$$

One can think of L as a list of parenthesized expressions, while N is a single, non-empty parenthesized expression. This is readily translated into an inductive definition via inference rules.

$$\frac{}{\varepsilon\ L} \qquad (4)$$

$$\frac{s_1\ N \qquad s_2\ L}{s_1\,s_2\ L} \qquad (5)$$

$$\frac{s\ L}{\texttt{(}s\texttt{)}\ N} \qquad (6)$$

Note that the definitions of $s\ L$ and $s\ N$ depend on each other. This is an example of a *simultaneous inductive definition*.

Now there are two important questions to ask: (1) is the new grammar really equivalent to the old one in the sense that it generates the same set of strings, and (2) is the new grammar really unambiguous. The latter is left as a (non-trivial!) exercise; the first one we discuss here.

At a high level we want to show that for any string s, $s\ M$ iff $s\ L$. We break this down into two lemmas. This is because "if-and-only-if" statement can rarely be proven by a single induction, but require different considerations for the two directions.

We first consider the direction where we assume $s\ M$ and try to show $s\ L$. When writing out the cases we notice we need an additional lemma. As is often the case, the presentation of the proof is therefore different from its order of discovery. To read this proof in a more natural order, skip ahead to Lemma 3 and pay particular attention to the last step in the case of rule (2). That step motivates the following lemma.

Lemma 2 (Concatenation)
If $s_1\ L$ and $s_2\ L$ then $s_1\, s_2\ L$.

Proof: By induction on the derivation of $s_1\ L$. Note that induction on the derivation on $s_2\ L$ will not work in this case!

(Rule 4) Then $s_1 = \varepsilon$.

$s_2\ L$	Assumption
$s_1\, s_2\ L$	Since $s_1\, s_2 = \varepsilon\, s_2 = s_2$

(Rule 5) Then $s_1 = s_{11}\, s_{12}$.

$s_{11}\ N$	Subderivation
$s_{12}\ L$	Subderivation
$s_2\ L$	Assumption
$s_{12}\, s_2\ L$	By i.h.
$s_{11}\, s_{12}\, s_2\ L$	By rule (5)

■

Now we are ready to prove the left-to-right implication.

Lemma 3
If $s\ M$ then $s\ L$.

Proof: By induction on the derivation of $s\ M$.

(Rule 1) Then $s = \varepsilon$.

$s\ L$	By rule (4) since $s = \varepsilon$

(Rule 2) Then $s = s_1\, s_2$.

$s_1\ M$	Subderivation
$s_2\ M$	Subderivation
$s_1\ L$	By i.h.
$s_2\ L$	By i.h.
$s_1\, s_2\ L$	By concatenation (Lemma 2)

(Rule 3) Then $s = \mathtt{(}s'\mathtt{)}$.

$s'\ M$	Subderivation
$s'\ L$	By i.h.
$\mathtt{(}s'\mathtt{)}\ N$	By rule (6)
$\varepsilon\ L$	By rule (4)
$\mathtt{(}s'\mathtt{)}\ L$	By rule (5) and $\mathtt{(}s'\mathtt{)}\ \varepsilon = \mathtt{(}s'\mathtt{)}$

■

The right-to-left direction presents a slightly different problem, namely that the statement "*If $s\ L$ then $s\ M$*" does not speak about $s\ N$, even though L and N depend on each other. In such a situation we typically have to generalize the induction hypothesis to also assert an appropriate property of the auxiliary judgments ($s\ N$, in this case). This is the first alternative proof below. The second alternative proof uses a proof principle called inversion, closely related to induction. We present both proofs to illustrate both techniques.

Lemma 4 (First Alternative, Using Generalization)

1. *If $s\ L$ then $s\ M$.*
2. *If $s\ N$ then $s\ M$.*

Proof: By simultaneous induction on the given derivations. There are two cases to consider for part 1 and one case for part 2.

(Rule 4) Then $s = \varepsilon$.

$s\ M$	By rule (1) since $s = \varepsilon$

(Rule 5) Then $s = s_1\, s_2$.

$s_1\ N$	Subderivation
$s_2\ L$	Subderivation
$s_1\ M$	By i.h.(2)
$s_2\ M$	By i.h.(1)
$s_1\, s_2\ M$	By rule (2)

(Rule 6) Then $s =$ `(`s'`)`.

$s'\ L$	Subderivation
$s'\ M$	By i.h.(1)
`(`s'`)` M	By rule (3)

■

For this particular lemma, we could have avoided the generalization and instead proven (1) directly by using a new form of argument called *inversion*. It is called inversion because it allows us to reason from the conclusion of an inference rule to its premises, while normally an inference rule works from the premises to the conclusion. This is confusing (and often applied incorrectly), so make sure you understand why and when this is legal by carefully examining the following proof.

Lemma 4 (Second Alternative, Using Inversion)
If $s\ L$ then $s\ M$

Proof: By induction on the given derivation. Note the there are only two cases to consider here instead of three, because there are only two rules whose conclusion has the form $s\ L$.

(Rule 4) Then $s = \varepsilon$.

$s\ M$	By rule (1) since $s = \varepsilon$

(Rule 5) Then $s = s_1\, s_2$.

$s_1\ N$	Subderivation
$s_1 =$ `(`s_1'`)` and $s_1'\ L$ for some s_1'	By inversion
$s_1'\ M$	By i.h.
`(`s_1'`)` M	By rule (3)
$s_2\ L$	Subderivation

$s_2\ M$ By i.h.
$(\ s_1'\ )\ s_2\ M$ By rule (2)
$s\ M$ Since $s = s_1\, s_2 = (\ s_1'\ )\ s_2$

In this last case, the first line reminds us that we have a subderivation of $s_1\ N$. By examining all inference rules we can see that there is exactly one rule that has a conclusion of this form, namely rule (6). Therefore $s_1\ N$ *must* have been inferred with that rule, and s_1 must be equal to $(\ s_1'\ )$ for some s_1' such that $s_1'\ L$. Moreover, the derivation of $s_1'\ L$ is a subderivation of the one we started with and we can therefore apply the induction hypothesis to it. The rest of the proof is routine. ∎

Now we can combine the preceding lemmas into the theorem we were aiming for.

Theorem 5
$s\ M$ if and only if $s\ L$.

Proof: Immediate from Lemmas 3 and 4. ∎

Some advice on inductive proofs. Most of the proofs that we will carry out in the class are by induction. This is simply due to the nature of the objects we study, which are generally defined inductively. Therefore, when presented with a conjecture that does not follow immediately from some lemmas, we first try to prove it by induction as given. This might involve a choice among several different given objects or derivations over which we may apply induction. If one of them works we are, of course, done. If not, we try to analyse the failure in order to decide if (a) we need to seperate out a *lemma* to be proven first, (b) we need to *generalize the induction hypothesis*, or (c) our conjecture might be false and we should look for a *counterexample*.

Finding a lemma is usually not too difficult, because it can be suggested by the gap in the proof attempt you find it impossible to fill. For example, in the proof of Lemma 3, case (Rule 2), we obtain $s_1\ L$ and $s_2\ L$ by induction hypothesis and have to prove $s_1\, s_2\ L$. Since there are no inference rules that would allow such a step, but it seems true nonetheless, we prove it as Lemma 2.

Generalizing the induction hypothesis can be a very tricky balancing act. The problem is that in an inductive proof, the property we are trying to establish occurs twice: once as an inductive assumption and once as a conclusion we are trying to prove. If we strengthen the property, the induction hypothesis gives us more information, but conclusion becomes harder to prove. If we weaken the property, the induction hypothesis gives us less information, but the conclusion is easier to prove. Fortunately, there are easy cases such as the first alternative of Lemma 4 in which the nature of the mutually recursive judgments suggested a generalization.

Finding a counterexample greatly varies in difficulty. Mostly, in this course, counterexamples only arise if there are glaring deficiencies in the inductive definitions, or rather obvious failure of properties such as type safety. In other cases it might require a very deep insight into the nature of a particular inductive definition and cannot be gleaned directly from a failed proof attempt. An example of a difficult counterexample is given by the extra credit Question 2.2 in Assignment 1 of this course. The conjecture might be that every tautology is a theorem. However, there is very little in the statement of this theorem or in the definition of *tautology* and *theorem* which would suggest means to either prove or refute it.

Three pitfalls to avoid. The difficulty with inductive proofs is that one is often blinded by the fact that the proposed conjecture is true. Similarly, if set up correctly, it will be true that in each case the induction hypothesis does in fact imply the desired conclusion, but the induction hypothesis may not be strong enough to prove it. So you must avoid the temptation to declare something as "clearly true" and prove it instead.

The second kind of mistake in an inductive proof that one often encounters is a confusion about the direction of an inference rule. If you reason backwards from what you are trying to prove, you are thinking about the rules bottom up: "*If I only could prove J_1 then I could conclude J_2, because I have an inference rule with premise J_1 and conclusion J_2.*" Nonetheless, when you write down the proof in the end you must use the rule in the proper direction. If you reason forward from your assumptions using the inference rules top-down then no confusion can arise. The only exception is the proof principle of inversion, which you can *only* employ if (a) you have established that a derivation of a given judgment J exists, and (b) you consider all possible inference rules whose conclusion matches J. In no other case can you use an inference rule "backwards".

The third mistake to avoid is to apply the induction hypothesis to a derivation that is not a subderivation of the one you are given. Such reasoning is circular and unsound. You must always verify that when you claim something follows by induction hypothesis, it is in fact legal to apply it!

How much to write down. Finally, a word on the level of detail in the proofs we give and the proofs we expect you to provide in the homework assignments. The proofs in this handout are quite pedantic, but we ask you to be just as pedantic unless otherwise specified. In particular, you *must* show any lemmas you are using, and you *must* show the generalized induction hypothesis in an inductive proof (if you need a generalization). You also *must* consider all the cases and *justify each line* carefully. As we gain a certain facility with such proofs, we may relax these requirements once we are certain you know how to fill in the steps that one might omit, for example, in a research paper.

Lecture 2 Addendum: A Parser in Judgment Form

During lecture, we also discussed that it is possible to write down a parser for the language of matching parentheses in the form of a judgment. The informal idea of the parsing process for matching parentheses is quite straightforward: we keep an integer counter, initialized to zero, and increment it when we see an opening parenthesis and decrement it when we see a closing parenthesis. We need to check two conditions: (a) the counter never becomes negative (otherwise there would be too many closing parentheses) and (b) the counter is zero at the end (otherwise there would be unmatched open parentheses).

Instead of an integer counter, our parser maintains a stack of open parentheses, where the stack is represented just as a string. The process of parsing corresponds to the bottom-construction of a derivation for a judgment

$$k \rhd s$$

which means that s is a valid string with respect to stack k. The symbol $\rhd$ has no special meaning here—it is simply used to separate the stack k from the string s. We now develop the rules for this two-place (binary) judgment.

First, if the string s is empty then we accept if the stack is also empty. This corresponds to condition (b) mentioned above.

$$\frac{}{\varepsilon \rhd \varepsilon}\ \rhd_1$$

Second, if the string s starts with an opening parenthesis, we push it on the stack. A less operational reading is: if s is a valid string in stack $k\,\texttt{(}$, then $\texttt{(}\,s$ is a valid string in stack k.

$$\frac{k\ \texttt{(}\ \rhd s}{k \rhd\ \texttt{(}\,s}\ \rhd_2$$

Finally, if we see a closing parenthesis at the beginning of the string, we pop the corresponding opening parenthesis from the stack and continue. A less operational reading is: if s is a valid string in stack k then $\texttt{)}\,s$ is a valid string in stack $k\,\texttt{(}$.

$$\frac{k \rhd s}{k\ \texttt{(}\ \rhd\ \texttt{)}\ s}\ \rhd_3$$

Since these are all the rules, the bottom-up construction of a derivation will get stuck if the string s begins with a closing parentheses but the stack

k is empty. That is, there is no rule with which we could infer $\varepsilon \rhd \texttt{)}\, s$, no matter what s is. This corresponds to condition (a) mentioned at the beginning of this discussion.

It is easy to see that this parser is inherently unambiguous. That is, when we start to construct a derivation of $\varepsilon \rhd s$ in order to parse s, then there is at most one rule that can be applied, depending on whether s is empty (rule $\rhd_1$), starts with an opening parenthesis (rule $\rhd_2$), or starts with a closing parenthesis (rule $\rhd_3$). Therefore, we can think of the judgment as describing a deterministic algorithm for parsing a string. This judgment can be related to a *push-down automaton*. As an aside, it turns out that every context-free grammar can be accepted by a (possibly non-deterministic) pushdown automaton, although the general construction of a pushdown automaton from a context-free grammar is more complex than in this particular example.

But does the judgment above really accept the language of properly balanced parentheses? We would like to prove that $s\ M$ if and only if $\varepsilon \rhd s$. As usual, we break this up into two separate lemmas, one for each direction.

For the first direction, we need one further lemma[1] that captures the essence of the left-to-right processing of the input string and the use of k as a stack.

Lemma 6 (Stack)
If $k_1 \rhd s_1$ *and* $k_2 \rhd s_2$ *then* $k_2\, k_1 \rhd s_1\, s_2$

Proof: By rule induction on the derivation of $k_1 \rhd s_1$.

(Rule $\rhd_1$) Then $k_1 = s_1 = \varepsilon$

$k_2 \rhd s_2$	Assumption
$k_2\, k_1 \rhd s_1\, s_2$	Since $k_1 = s_1 = \varepsilon$.

(Rule $\rhd_2$) Then $s_1 = \texttt{(}\, s_1'$.

$k_1\, \texttt{(} \rhd s_1'$	Subderivation
$k_2 \rhd s_2$	Assumption
$k_2\, k_1\, \texttt{(} \rhd s_1'\, s_2$	By i.h.
$k_2\, k_1 \rhd \texttt{(}\, s_1'\, s_2$	By rule ($\rhd_2$)

[1] suggested by a student during lecture

(Rule $\triangleright_3$) Then $k_1 = k_1'$ `(` and $s_1 =$ `)` s_1'.

$k_1' \triangleright s_1'$	Subderivation
$k_2 \triangleright s_2$	Assumption
$k_2\, k_1' \triangleright s_1'\, s_2$	By i.h.
$k_2\, k_1'$ `(` $\triangleright$ `)` $s_1'\, s_2$	By rule ($\triangleright_3$)

∎

Now we can prove the first direction of the correctness theorem for the parser.

Lemma 7
If $s\ M$ then $\varepsilon \triangleright s$.

Proof: By rule induction on the derivation of $s\ M$.

(Rule 1) Then $s = \varepsilon$.

$\varepsilon \triangleright \varepsilon$	By rule ($\triangleright_1$)

(Rule 2) Then $s = s_1\, s_2$.

$s_1\ M$	Subderivation
$\varepsilon \triangleright s_1$	By i.h.
$s_2\ M$	Subderivation
$\varepsilon \triangleright s_2$	By i.h.
$\varepsilon \triangleright s_1\, s_2$	By Lemma 6

(Rule 3) Then $s =$ `(`s'`)`.

$s'\ M$	Subderivation
$\varepsilon \triangleright s'$	By i.h.
$\varepsilon \triangleright \varepsilon$	By rule ($\triangleright_1$)
`(` $\triangleright$ `)`	By rule ($\triangleright_3$)
`(` $\triangleright s'$`)`	By Lemma 6
$\varepsilon \triangleright$ `(`s'`)`	By rule ($\triangleright_2$)

∎

In order to prove the other direction (if $\varepsilon \triangleright s$ then $s\ M$) we first generalize to: *If $k \triangleright s$ then $k\, s\ M$.* This proof (which is left to the reader) requires another lemma, this time about the M judgment.

Finally, putting the two directions together proves the correctness of our parser.