1 Reading

- EOPL 3rd Edition, Chapter 1

2 Inductive Specifications

The big picture: languages contain infinite number of sentences, so must be inductively specified. Grammars are a way of specifying inductive sets. Interpreters are recursive because they follow grammar. For us, inductive specifications are more important than induction as a proof technique.

2.1 What are induction specifications?

An inductive specification is useful for saying what the values are in an ADT or a programming language. Following are some examples of inductive specifications.

Definition 2.1 Set of Natural Numbers: the set $N$ of natural numbers is the smallest set such that:

1. 0 is in $N$
2. if $i$ in $N$, then $i+1$ is in $N$

Exercise 2.1 Why isn’t $N = \ldots, -2, -1, 0, 1, 2, \ldots$?

It is not the smallest set. See definition of smallest set in EOPL page 3. Try suggested exercise 1.3.

Definition 2.2 Set of Numerals: Let Digit = \{0,1,2,3,4,5,6,7,8,9\}. The set Numeral of numerals is the smallest set such that:

1. if $x$ is in Digit, then $x$ in Numeral
2. if $x$ is in Digit, and $n$ in Numeral, then $x:n$ is in Numeral.

Exercise 2.2 Let symbol be the set of symbols. Write an inductive specification of the set (list-of symbol), i.e., the set of lists of symbols.

The set (list-of symbol) of lists of symbols is the smallest set such that

1. () is in (list-of symbol)
2. if $s$ is a symbol, and $l$ is a (list-of symbol), then the pair (s . l) is a (list-of symbol).

Exercise 2.3 Let symbol be the set of symbols. Give an alternative inductive specification of the set (list-of symbol), i.e., the set of lists of symbols.

The following is an alternative inductive definition: if (s1 ...) is a (list-of symbol) then (s s1 ...) is a (list-of symbol).
Exercise 2.4  Let symbol be the set of symbols. Write a NON-inductive specification of the set (list-of symbol), i.e., the set of lists of symbols.

The following isn’t inductive: if s1, ..., sn are symbols, for n>=0, then (s1 ... sn) is a (list-of symbol)

An alternative method for inductive specifications is via rules of inferences (see EOPL, page 3) and learn about the terms hypothesis, conclusion, and axiom in this context.

3  The Use of Grammars in Programming Languages

In definition:

all strings over alphabet  
regular grammar  
context-free grammar  

In implementation (parsing):

input tokens  
 lexer  parser  
 trees  
 code gen  
 code  
 object module

Exercise 3.5  Why do programming languages typically use 2 layers of grammar?

It allows more abstraction.

Exercise 3.6  How could we avoid the repetition in inductive specifications?

Use a special notation, grammars...
4 Backus-Naur Form (BNF) (EOPL 1.1.2)

\[
\text{list-of-symbol} ::= ( )
\]

The grammar above can also be written in the following alternative shorthand form.

\[
\text{list-of-symbol} ::= ( )
| ( \text{symbol} \cdot \text{list-of-symbol} )
\]

Notation, terms (read definitions from EOPL, Section 1.1.2)

- ::= 
- syntactic category, non-terminal 
- terminal 
- rule, production 
- separated list notation

Exercise 4.7 [For You to Do in Pairs] Given \text{symbol} and \text{number}, write a grammar for \text{expression} that includes the following examples:
3
x
(5 + 4)
(x + ((7 * 8) / y))
(9 - z)

\[
\text{expression} ::= \text{number} \mid \text{symbol}
| (\text{expression} \ \text{op} \ \text{expression})
\]

\[
\text{op} ::= + \mid - \mid * \mid /
\]

5 Kleene Star and Plus

Kleene star and plus are generally used in regular grammars. Much of this is explained in your textbook. An example usage for these is shown below.

\[
\text{list-of-symbol} ::= ( \{\text{symbol}\}* )
\]

\[
\text{application} ::= ( \{\text{expression}\}+ )
\]

A way to think about Kleene star and plus is to replace them with the following implicit grammar rules:

\[
\{X\}^* ::= \mid \{X\}^* \\
\{X\}^+ ::= \mid \{X\}^+
\]

Let us look at another example usage.

\[
\text{digit} ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\]

\[
\text{number} ::= \{\text{digit}\}+
\]

\[
\text{letter} ::= a \mid \ldots \mid z \mid A \mid \ldots \mid Z
\]

\[
\text{idchar} ::= - \mid _ \mid + \mid * \mid ? \mid / \mid ! \mid \text{letter}
\]

\[
\text{idOrN} ::= \text{idchar} \mid \text{number}
\]

\[
\text{symbol} ::= \{\text{idOrN}\}^* \text{idchar} \{\text{idOrN}\}^*
\]
6 Derivations

A derivation is the process of replace one non-terminal using a rule at each step. To illustrate, let us consider the following grammar for the list of symbols.

\[\text{<list-of-symbol> ::= ( ) | ( <symbol> . <list-of-symbol> )}\]

Derivation of \(\text{fly}\):

\[
\begin{align*}
\text{<list-of-symbol>} &\Rightarrow ( <symbol> . <list-of-symbol> ) \\
&\Rightarrow ( \text{fly} . <list-of-symbol> ) \\
&\Rightarrow ( \text{fly} . () ) \\
&= ( \text{fly} ) /* because of the equivalence of ( fly . () ) and (fly) */
\end{align*}
\]

The following illustrates the same derivation, but now using the implicit rules for Kleene Star. The implicit rules used are:

\[
\begin{align*}
\{<X>\}^+ &\Rightarrow <X> \{<X>\}^+ \\
\{<X>\}^+ &\Rightarrow <X> \{<X>\}^+
\end{align*}
\]

Grammar:

\[\text{<list-of-symbol> ::= ( \{<symbol>\}* )}\]

Derivation of \(\text{fly}\):

\[
\begin{align*}
\text{<list-of-symbol>} &\Rightarrow ( \{<symbol>\}* ) \\
&\Rightarrow ( <symbol> \{<symbol>\}* ) \\
&\Rightarrow ( \text{fly} \{<symbol>\}* ) \\
&\Rightarrow ( \text{fly} )
\end{align*}
\]

Exercise 6.8  How does this help in defining a programming language?

Exercise 6.9  [For You To Do in Pairs] Consider the following grammar:

\[\text{<expression> ::= <number> | <identifier> | ( \lambda \{<identifier>\}^+ ) <expression> | \{<expression>\}^+}\]

If the following are \(<expression>\)s, then give a derivation, else show why there is no derivation.

1. \(x\)
2. \((\lambda (x) \, x) \, 3)\)
3. \(f(3, 4)\)

Note that the implicit rule used here is:

\[
\begin{align*}
\{<X>\}^+ &\Rightarrow <X> \\
\{<X>\}^+ &\Rightarrow <X> \{<X>\}^+
\end{align*}
\]
6.1 Summary

BNF is a good way to write an inductive specification, especially for a language.

7 Using BNF to specify data (EOPL 2.1.3)

\[
\text{<list-of-symbol>} ::= \{ \text{<symbol>}\}^* \\
\text{<list-of-symbol>} ::= \{ \} \\
\quad | \quad \{ \text{<symbol>} \cdot \text{<list-of-symbol>} \}
\]

\[
\text{<closure>} ::= \{ \text{<code>} \cdot \text{<environment>} \}
\text{<code>} ::= \{ \text{<formals>} \cdot \text{<expression>} \}
\text{<environment>} ::= \{ \text{<parent-env>} \cdot \{ \{\text{<binding>}\}^* \}
\text{<parent-env>} ::= () | \text{<environment>}
\text{<binding>} ::= \{ \text{<var>} \cdot \text{<datum>} \}
\]

\[
\text{<statement>} ::= \{ \text{exp-stmt} \cdot \text{<expression>} \}
\quad | \quad \{ \text{set!} \cdot \text{<identifier>} \cdot \text{<expression>} \}
\text{<expression>} ::= \{ \text{var-exp} \cdot \text{<identifier>} \}
\quad | \quad \{ \text{num-exp} \cdot \text{<number>} \}
\quad | \quad \{ \text{begin} \cdot \{\text{<statement>}\}^* \cdot \text{<expression>} \}
\]

Following are some important ideas for BNF:

- Different ways of specifying the same data are possible, as with the list-of-symbol example.
- The grammar can describe several data structures, as in the closure example.

Exercise 7.10 What would be the base case for a procedure that took a <list-of-symbol> as an argument?

Exercise 7.11 What would the recursive case be?

Exercise 7.12 What are the cases for a <closure>?

Exercise 7.13 What would the recursive case be?

Exercise 7.14 What are the cases for <statement>? <expression>?
8 Specifying Scheme Data

⟨datum⟩ ::= ⟨number⟩ | ⟨symbol⟩ | ⟨char⟩
    | ⟨boolean⟩ | ⟨string⟩
    | ⟨list-of-T⟩ | ⟨pair-of-S-T⟩
    | ⟨vector-of-T⟩
⟨list-of-T⟩ ::= ( {⟨T⟩}* )
⟨pair-of-S-T⟩ ::= ( ⟨S⟩ . ⟨T⟩ )
⟨vector-of-T⟩ ::= #{ {⟨T⟩}* }
⟨T⟩ ::= ⟨datum⟩
⟨S⟩ ::= ⟨datum⟩

Exercise 8.15  How would you specify ⟨list⟩ without using the Kleene star?

9 Context-free vs. Context-sensitive

See EOPL, page 10 for more information and examples that illustrate difference between context-sensitive and context-free languages.

⟨bin-tree⟩ ::= () | (⟨key⟩ ⟨bin-tree⟩ ⟨bin-tree⟩)
⟨key⟩ ::= ⟨number⟩

Exercise 9.16  What would make this into a binary search tree?

Exercise 9.17  Is there any way to specify this in a context-free grammar?

Exercise 9.18  What are context-sensitive constraints? or invariants?