2 Programming Language Syntax

2.4 Theoretical Foundations

As noted in the main text, scanners and parsers are based on the finite automata and pushdown automata that form the bottom two levels of the Chomsky language hierarchy. At each level of the hierarchy, machines can be either *deterministic* or *nondeterministic*. A deterministic automaton always performs the same operation in a given situation. A nondeterministic automaton can perform any of a *set* of operations. A nondeterministic machine is said to accept a string if there exists a choice of operation in each situation that will eventually lead to the machine saying "yes." As it turns out, nondeterministic and deterministic finite automata are equally powerful: every DFA is, by definition, a degenerate NFA, and the construction in Example 2.14 demonstrates that for any NFA we can create a DFA that accepts the same language. The same is not true of push-down automata: there are context-free languages that are accepted by an NPDA but not by any DPDA. Fortunately, DPDAs suffice in practice to accept the syntax of real programming languages. Practical scanners and parsers are always deterministic.

2.4.1 Finite Automata

Precisely defined, a deterministic finite automaton (DFA) M consists of (1) a finite set Q of *states*, (2) a finite alphabet Σ of input symbols, (3) a distinguished *initial* state $q_1 \in Q$, (4) a set of distinguished *final* states $F \subseteq Q$, and (5) a *transition function* $\delta : Q \times \Sigma \rightarrow Q$ that chooses a new state for M based on the current state and the current input symbol. M begins in state q_1. One by one it consumes its input symbols, using δ to move from state to state. When the final symbol has been consumed, M is interpreted as saying "yes" if it is in a state in F; otherwise it is interpreted as saying "no." We can extend δ in the obvious way to take strings, rather than symbols, as inputs, allowing us to say that M accepts string x if $\delta(q_1, x) \in F$. We can then define $L(M)$, the language accepted by M, to be the set

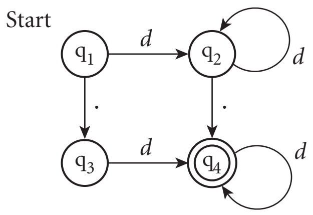

Figure 2.33 **Minimal DFA for the language consisting of all strings of decimal digits containing a single decimal point.** Adapted from Figure 2.10 in the main text. The symbol d here is short for "0, 1, 2, 3, 4, 5, 6, 7, 8, 9".

$\{x \mid \delta(q_1, x) \in F\}$. In a nondeterministic finite automaton (NFA), the transition function, δ, is multivalued: the automaton can move to any of a *set* of possible states from a given state on a given input. In addition, it may move from one state to another "spontaneously"; such transitions are said to take input symbol ϵ.

EXAMPLE 2.56
Formal DFA for d*(. d | d .) d*

We can illustrate these definitions with an example. Consider the circles-and-arrows automaton of Figure C-2.33 (adapted from Figure 2.10 in the main text). This is the minimal DFA accepting strings of decimal digits containing a single decimal point. $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{2}, \texttt{3}, \texttt{4}, \texttt{5}, \texttt{6}, \texttt{7}, \texttt{8}, \texttt{9}, \texttt{.}\}$ is the machine's input alphabet. $Q = \{q_1, q_2, q_3, q_4\}$ is the set of states; q_1 is the initial state; $F = \{q_4\}$ (a singleton in this case) is the set of final states. The transition function can be represented by a set of triples $\delta = \{(q_1, \texttt{0}, q_2), \ldots, (q_1, \texttt{9}, q_2), (q_1, \texttt{.}, q_3), (q_2, \texttt{0}, q_2), \ldots, (q_2, \texttt{9}, q_2), (q_2, \texttt{.}, q_4), (q_3, \texttt{0}, q_4), \ldots, (q_3, \texttt{9}, q_4), (q_4, \texttt{0}, q_4), \ldots, (q_4, \texttt{9}, q_4)\}$. In each triple $(q_i, \texttt{a}, q_j)$, $\delta(q_i, \texttt{a}) = q_j$. ■

Given the constructions of Examples 2.12 and 2.14, we know that there exists an NFA that accepts the language generated by any given regular expression, and a DFA equivalent to any given NFA. To show that regular expressions and finite automata are of equivalent expressive power, all that remains is to demonstrate that there exists a regular expression that generates the language accepted by any given DFA. We illustrate the required construction below for our decimal strings example (Figure C-2.33). More formal and general treatment of all the regular language constructions can be found in standard automata theory texts [HMU07, Sip13].

From a DFA to a Regular Expression

To construct a regular expression equivalent to a given DFA, we employ a dynamic programming algorithm that builds solutions to successively more complicated subproblems from a table of solutions to simpler subproblems. We begin with a set of simple regular expressions that describe the transition function, δ. For all states i, we define

$$r_{ii}^0 = \texttt{a}_1 \mid \texttt{a}_2 \mid \ldots \mid \texttt{a}_m \mid \epsilon$$

where $\{\mathtt{a}_1 \mid \mathtt{a}_2 \mid \dots \mid \mathtt{a}_m\} = \{\mathtt{a} \mid \delta(q_i, \mathtt{a}) = q_i\}$ is the set of characters labeling the "self-loop" from state q_i back to itself. If there is no such self-loop, $r^0_{ij} = \epsilon$. Similarly, for $i \neq j$, we define

$$r^0_{ij} = \mathtt{a}_1 \mid \mathtt{a}_2 \mid \dots \mid \mathtt{a}_m$$

where $\{\mathtt{a}_1 \mid \mathtt{a}_2 \mid \dots \mid \mathtt{a}_m\} = \{\mathtt{a} \mid \delta(q_i, \mathtt{a}) = q_j\}$ is the set of characters labeling the arc from q_i to q_j. If there is no such arc, r^0_{ij} is the empty regular expression. (Note the difference here: we can stay in state q_i by not accepting any input, so ϵ is always one of the alternatives in r^0_{ii}, but not in r^0_{ij} when $i \neq j$.)

Given these r^0 expressions, the dynamic programming algorithm inductively calculates expressions r^k_{ij} with larger superscripts. In each, k names the highest-numbered state through which control may pass on the way from q_i to q_j. We assume that states are numbered starting with q_1, so when $k = 0$ we must transition directly from q_i to q_j, with no intervening states.

EXAMPLE 2.57

Reconstructing a regular expression for the decimal string DFA

In our small example DFA, $r^0_{11} = r^0_{33} = \epsilon$, and $r^0_{22} = r^0_{44} = \mathtt{0} \mid \mathtt{1} \mid \mathtt{2} \mid \mathtt{3} \mid \mathtt{4} \mid \mathtt{5} \mid \mathtt{6} \mid \mathtt{7} \mid \mathtt{8} \mid \mathtt{9} \mid \epsilon$, which we will abbreviate $d \mid \epsilon$. Similarly, $r^0_{13} = r^0_{24} = \mathtt{.}$, and $r^0_{12} = r^0_{34} = d$. Expressions $r^0_{14}, r^0_{21}, r^0_{23}, r^0_{31}, r^0_{32}, r^0_{41}, r^0_{42}$, and r^0_{43} are all empty.

For $k > 0$, the r^k_{ij} expressions will generally generate multicharacter strings. At each step of the dynamic programming algorithm, we let

$$r^k_{ij} = r^{k-1}_{ij} \mid r^{k-1}_{ik} r^{k-1}_{kk}{}^{\star} r^{k-1}_{kj}$$

That is, to get from q_i to q_j without going through any states numbered higher than k, we can either go from q_i to q_j without going through any state numbered higher than $k-1$ (which we already know how to do), or else we can go from q_i to q_k (without going through any state numbered higher than $k-1$), travel out from q_k and back again an arbitrary number of times (never visiting a state numbered higher than $k-1$ in between), and finally go from q_k to q_j (again without visiting a state numbered higher than $k-1$). If any of the constituent regular expressions is empty, we omit its term of the outermost alternation. At the end, our overall answer is $r^n_{1f_1} \mid r^n_{1f_2} \mid \dots \mid r^n_{1f_t}$, where $n = |Q|$ is the total number of states and $F = \{q_{f_1}, q_{f_2}, \dots, q_{f_t}\}$ is the set of final states.

Because $r^0_{11} = \epsilon$ and there are no transitions from States 2, 3, or 4 to State 1, nothing changes in the first inductive step in our example; that is, $\forall i\, [r^1_{ii} = r^0_{ii}]$. The second step is a bit more interesting. Since we are now allowed to go through State 2, we have $r^2_{22} = r^2_{22}\ r^2_{22}{}^{\star} r^2_{22} = (\,d \mid \epsilon\,) \mid (\,d \mid \epsilon\,)\,(\,d \mid \epsilon\,)^{\star}(\,d \mid \epsilon\,) = d^{\star}$. Because $r^1_{21}, r^1_{23}, r^1_{32}$, and r^1_{42} are empty, however, r^2_{11}, r^2_{33}, and r^2_{44} remain the same as r^1_{11}, r^1_{33}, and r^1_{44}. In a similar vein, we have

$$\begin{aligned} r^2_{12} &= d \mid d\,(\,d \mid \epsilon\,)^{\star}(\,d \mid \epsilon\,) \;=\; d^{+} \\ r^2_{14} &= d\,(\,d \mid \epsilon\,)^{\star}\,\mathtt{.} \;=\; d^{+}\,\mathtt{.} \\ r^2_{24} &= \mathtt{.} \mid (\,d \mid \epsilon\,)\,(\,d \mid \epsilon\,)^{\star}\,\mathtt{.} \;=\; d^{\star}\,\mathtt{.} \end{aligned}$$

Missing transitions and empty expressions from the previous step leave $r^2_{13} = r^1_{13}$ = . and $r^2_{34} = r^1_{34} = d$. Expressions $r^2_{21}, r^2_{23}, r^2_{31}, r^2_{32}, r^2_{41}, r^2_{42}$, and r^2_{43} remain empty.

In the third inductive step, we have

$$\begin{aligned} r^3_{13} &= \;.\; \mid \;.\; \epsilon^\star \epsilon \;=\; . \\ r^3_{14} &= d^+ \;.\; \mid \;.\; \epsilon^\star d \;=\; d^+ \;.\; \mid \;.\; d \\ r^3_{34} &= d \mid \epsilon\epsilon^\star d \;=\; d \end{aligned}$$

All other expressions remain unchanged from the previous step.

Finally, we have

$$\begin{aligned} r^4_{14} &= (\, d^+ \;.\; \mid \;.\; d \,) \mid (\, d^+ \;.\; \mid \;.\; d \,)\,(\, d \mid \epsilon \,)^\star (\, d \mid \epsilon \,) \\ &= (\, d^+ \;.\; \mid \;.\; d \,) \mid (\, d^+ \;.\; \mid \;.\; d \,)\; d^\star \\ &= (\, d^+ \;.\; \mid \;.\; d \,)\; d^\star \\ &= d^+ \;.\; d^\star \mid \;.\; d^+ \end{aligned}$$

Since F has a single member (q_4), this expression is our final answer. ■

Space Requirements

In Section 2.2.1 we noted without proof that the conversion from an NFA to a DFA may lead to exponential blow-up in the number of states. Certainly this did not happen in our decimal string example: the NFA of Figure 2.8 has 14 states, while the equivalent DFA of Figure 2.9 has only 7, and the minimal DFA of Figures 2.10 and C-2.33 has only 4.

EXAMPLE 2.58
A regular language with a large minimal DFA

Consider, however, the subset of `( a | b | c )*` in which some letter appears at least three times. The minimal DFA for this language has 28 states. As shown in Figure C-2.34, 27 of these are states in which we have seen i, j, and k `a`s, `b`s, and `c`s, respectively. The 28th (and only final) state is reached once we have seen at least three of some specific character.

By contrast, there exists an NFA for this language with only eight states, as shown in Figure C-2.35. It requires that we "guess," at the outset, whether we will see three `a`s, three `b`s, or three `c`s. It mirrors the structure of the natural regular expression `( a | b | c )* a ( a | b | c )* a ( a | b | c )* a ( a | b | c )* | ( a | b | c )* b ( a | b | c )* b ( a | b | c )* b ( a | b | c )* | ( a | b | c )* c ( a | b | c )* c ( a | b | c )* c ( a | b | c )*`. ■

EXAMPLE 2.59
Exponential DFA blow-up

Of course, the eight-state NFA does not emerge directly from the construction of Figure 2.7; that construction produces a 52-state machine with a certain amount of redundancy, and with many extraneous states and epsilon productions. But consider the similar subset of `( 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 )*` in which some digit appears at least ten times. The minimal DFA for this language has 10,000,000,001 states: a non-final state for each combination of zeros through nines with less than ten of each, and a single final state reached once any digit has appeared at least ten times. One possible regular expression for this language is

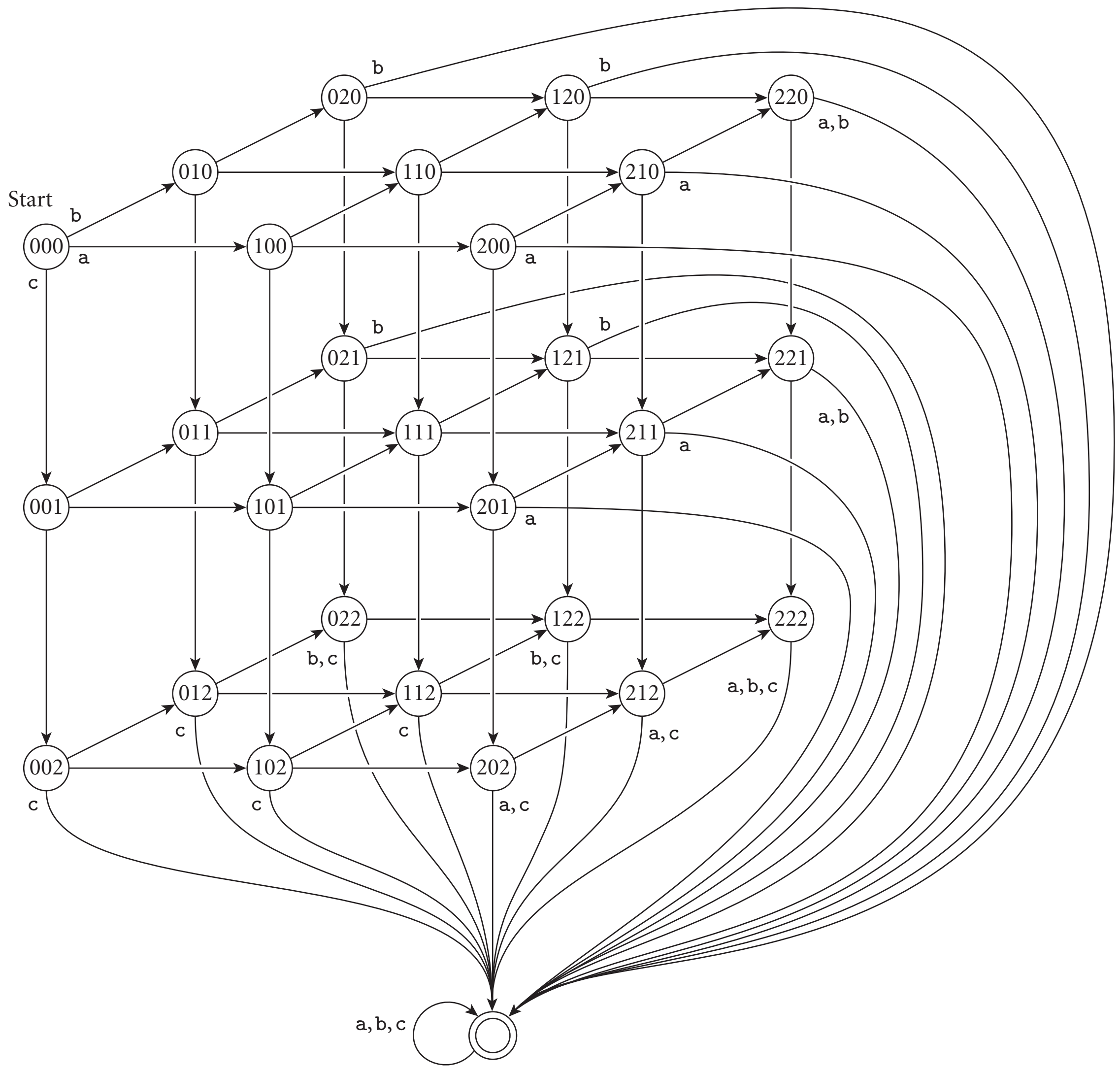

Figure 2.34 **DFA for the language consisting of all strings in (a | b | c)* in which some letter appears at least three times.** State name *ijk* indicates that *i* as, *j* bs, and *k* cs have been seen so far. Within the cubic portion of the figure, most edge labels are elided: a transitions move to the right, b transitions go back into the page, and c transitions move down.

```
 ((0|1|...|9)* 0 (0|1|...|9)* 0 (0|1|...|9)* 0 (0|1|...|9)* 0
   (0|1|...|9)* 0 (0|1|...|9)* 0 (0|1|...|9)* 0 (0|1|...|9)* 0
   (0|1|...|9)* 0 (0|1|...|9)* 0 (0|1|...|9)*)
|((0|1|...|9)* 1 (0|1|...|9)* 1 (0|1|...|9)* 1 (0|1|...|9)* 1
   (0|1|...|9)* 1 (0|1|...|9)* 1 (0|1|...|9)* 1 (0|1|...|9)* 1
   (0|1|...|9)* 1 (0|1|...|9)* 1 (0|1|...|9)*)
|  ...
|((0|1|...|9)* 9 (0|1|...|9)* 9 (0|1|...|9)* 9 (0|1|...|9)* 9
   (0|1|...|9)* 9 (0|1|...|9)* 9 (0|1|...|9)* 9 (0|1|...|9)* 9
   (0|1|...|9)* 9 (0|1|...|9)* 9 (0|1|...|9)*)
```

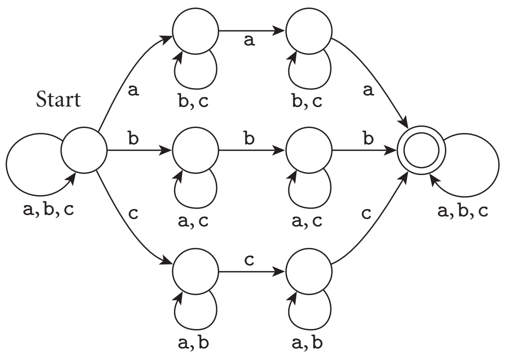

Figure 2.35 NFA corresponding to the DFA of Figure C-2.34.

Our construction would yield a very large NFA for this expression, but clearly many orders of magnitude smaller than ten billion states! ■

2.4.2 Push-Down Automata

A deterministic push-down automaton (DPDA) N consists of (1) Q, (2) Σ, (3) q_1, and (4) F, as in a DFA, plus (6) a finite alphabet Γ of stack symbols, (7) a distinguished initial stack symbol $Z_1 \in \Gamma$, and (5′) a transition function $\delta : Q \times \Gamma \times \{\Sigma \cup \{\epsilon\}\} \rightarrow Q \times \Gamma^*$, where Γ^* is the set of strings of zero or more symbols from Γ. N begins in state q_1, with symbol Z_1 in an otherwise empty stack. It repeatedly examines the current state q and top-of-stack symbol Z. If $\delta(q,\epsilon, Z)$ is defined, N moves to state r and replaces Z with α in the stack, where $(r, \alpha) = \delta(q,\epsilon, Z)$. In this case N does not consume its input symbol. If $\delta(q,\epsilon, Z)$ is undefined, N examines and consumes the current input symbol a. It then moves to state s and replaces Z with β, where $(s, \beta) = \delta(q, \mathtt{a}, Z)$. N is interpreted as accepting a string of input symbols if and only if it consumes the symbols and ends in a state in F.

As with finite automata, a nondeterministic push-down automaton (NPDA) is distinguished by a multivalued transition function: an NPDA can choose any of a set of new states and stack symbol replacements when faced with a given state, input, and top-of-stack symbol. If $\delta(q,\epsilon, Z)$ is nonempty, N can also choose a new state and stack symbol replacement without inspecting or consuming its current input symbol. While we have seen that nondeterministic and deterministic finite automata are equally powerful, this correspondence does not carry over to push-down automata: there are context-free languages that are accepted by an NPDA but not by any DPDA.

The proof that CFGs and NPDAs are equivalent in expressive power is more complex than the corresponding proof for regular expressions and finite automata. The proof is also of limited practical importance for compiler construction; we do not present it here. While it is possible to create an NPDA for any