Chapter 3 Scanning

The primary function of a scanner is to transform a character stream into a token stream. A scanner is sometimes called a lexical analyzer, or lexer. Programming a scanner generator is an example of declarative programming. We tell the program what to scan.

Regular expressions are a convenient they can specify the structure of the tokens used in a programming language. In particular, you can use regular expressions to program a scanner generator

- P+, sometimes called positive closure, denotes all strings consisting of one or more strings in P catenated together: P* = (P+| λ) and P+ = P P*.
- If A is a set of characters, Not(A) denotes (Σ - A), that is, all characters in Σ not included in A.
- If k is a constant, then the set Ak represents all strings formed by catenating k (possibly different) strings from A.

A finite automaton (FA) can be used to recognize the tokens specified by a regular expression: A finite set of states, A finite vocabulary, denoted Σ, A set of transitions (or moves) from one state to another, labeled with characters in Σ, A special state called the start state, A subset of the states called the accepting, or final, states.

An FA that always has a unique transition (for a given state and character) is a deterministic finite automaton (DFA). Its behavior can be represented in a transition table.

Case of study implementing RegEx, FDA, DFA, NDFA in Lex as an scanner…

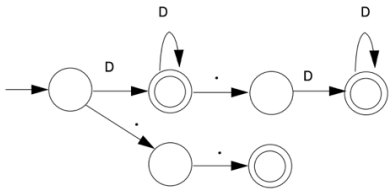

Figure 3.14: An FA that scans integer and real literals and the subrange operator.

Hence, it is a good idea to consider how to increase scanning speed (talking about performance) One approach to increasing scanner speed is to use a scanner generator such as Flex or GLA that is designed to generate fast scanners.

If you hand-code a scanner, a few general principles can increase scanner performance dramatically. Try to block character-level operations whenever possible. Read by buffers instead of chars. One problem with reading blocks of characters is that the end of a block won't usually correspond to the end of a token. Double-buffering can avoid this problem Input is first read into the left buffer, then into the right buffer, and then the left buffer is overwritten. Unless a token whose text we want to save is longer than the buffer length, tokens can cross a buffer boundary without difficulty

Regular expressions are equivalent to FAs. A scanner generator first creates an NFA from a set of regular-expression specifications. The NFA is then transformed into a DFA.

The transformation from an NFA N to an equivalent DFA D works by what is sometimes called the subset construction. The idea is that D will be in state {x, y, z} after reading a given input string if, and only if, N could be in any of the states x, y, or z.

Some DFAs contain unreachable states, states that cannot be reached from the start state. Other DFAs may contain dead states, states that cannot reach any accepting state, So we eliminate all such states as part of our optimization process.

Since regular expressions, DFAs, and NFAs are interconvertible, it is also possible to derive for any FA a regular expression that describes the strings that the FA matches