Programmation Avancée et Compilation 2001
(Solutions)

3 Generated Scanners

Keywords 3

Give short explanations of the following keywords:

> *Scanner generator*(Lexical Analysis 1, 14), *JLex*(Lexical Analysis 1, 15).

Exercise 4

(a) A binary number is made up of arbitrarily many 0s and 1s, yielding [01]+.

(b) In order to strip off leading 0s, we have to distinguish two cases: either the number is 0 or it is preceded by a 1; that is, we obtain 0|1[01]∗.

(c) In order to obtain an even number of 1s, we have to make sure that whenever a 1 appears after arbitrarily many other digits, there will be yet another one, again after arbitrarily many other digits, described by [02–9]∗1[02–9]∗1. Afterwards, the whole game can restart, yielding ([02–9]∗1[02–9]∗1)∗. Finally, we have to consider that there are also numbers ending with another digit than 1, and we obtain ([02–9]∗1[02–9]∗1)∗[02–9]∗. Note that we cannot apply [^1], because this would allow letters as well.

A language is regular, either if it contains only finitely many words, or else if the analysis of a word does not necessitate to record (that is, to store somewhere) arbitrarily long subsequences of it.

(d) A binary number satisfying our specification can either be 0 or 1, or can contain an arbitrary binary number between 0...0 or 1...1. This can be described as follows: 0|1|0[01]∗0|1[01]∗1.

(e) The language of palindromes is not regular, thus cannot be described by a JLex expression. As for an explanation (nothing you will have to reproduce), a palindrome is made up of two halves, possibly containing a letter in between, where the second half is a mirrored version of the first one; that is, we either have ww^{-1} or waw^{-1}. The w can be a string of arbitrary length; to describe this we need a ∗. But how can we tell which word should actually be derived in a specific case so to mirror it and copy it to the end? Exactly this cannot be described by any construct JLex gives us.

    Alternatively, considering the analysis of such words by a finite-state (!) automaton, one has to record the first half of the word somehow in order to map it to the second half of it. But the words, and hence also the halves, can be arbitrarily long. This can certainly not be managed if one can only use a finite number of states.

(f) By using `a*[^ab]` we can ensure that a sequence of `a`s is always terminated by something to which we can append whatever we like (if we write `a*[^b]`, we can still have an `a` at the end disallowing the use of a `b`). Otherwise, also anything else but an `a` (that is, `[^a]`) allows us to append whatever we like. This yields `(a*[^ab]|[^a])*`, as the pattern is safe and can therefore be repeated. Finally, we still have to consider the case where a word terminates with a sequence of `a`s, and we obtain `(a*[^ab]|[^a])*a*`. Note that `[^ab]*` yields sequences containing neither `a` nor `b`.