Scanner Specification

A scanner can be specified by defining the tokens used by the parser. The left-hand side of a token definition can be used as a terminal in the production rules of a parser specification. A token can be defined as a list of keyword or symbol alternatives or as a regular expression of strings and character sets. A keyword or symbol is a sequence of printable characters. The scanner generator converts the symbols and regular expressions into a non-deterministic finite automaton (NFA) using the element construction algorithm. This NFA contains no empty transitions. The NFA is then converted into a deterministic finite automaton (DFA) using the subset construction algorithm. During subset construction, state transition conflicts are reported. Finally, the number of states of the DFA are minimised, using refined partitioning, resulting in the minimised deterministic finite automaton (MDFA). The element construction, subset construction and refined partitioning algorithms can be found in Chapter 7 in [AN96]. The NFA, DFA and MDFA diagrams can be viewed in the *.lst* file if the corresponding scanner options are selected.

Unlike symbols and regular expressions, keywords are not converted into a NFA, they are stored in the keyword table (to reduce the number of NFA states). You will need to define an extra token to recognise all the keyword tokens. If a token is scanned and it satisfies the extra keyword token definition, the keyword table is searched and, if found, the corresponding token is passed through to the parser.

Syntactic operators

As mentioned above, tokens can be defined using regular expressions. The following syntactic operators can be used in regular expressions:

`{ A }`	for grouping a regular expression `A`,
`A B`	for concatenating expressions `A` and `B`,
`A | B`	for separating alternative expressions `A` and `B`,
`A` **`CLOS`**	for zero or more instances of an expression `A`,
`A` **`OPTION`**	for zero or one instance of an expression `A`,
`A` **`SEQ`**	for one or more instances of an expression `A`,
`A + B`	union of character sets `A` and `B`,
`A - B`	difference of character sets `A` and `B`,
`A * B`	intersection of character sets `A` and `B`.

Precedence

All operators are left-associative and:

1. the binary character set operators `+`, `-` and `*` have the highest precedence,
2. the unary operators **`OPTION`**, **`SEQ`** and **`CLOS`** have the second highest precedence,
3. concatenation has the third highest precedence, and
4. the alternative separator `|` has the lowest precedence.

Alternative specifications:

`A` **`CLOS`** can be rewritten as: `{ A` **`SEQ`** `}` **`OPTION`**

`A + B` can be rewritten as: `A | B`

Scanner attributes

The scanner generator allows you to associate synthesised attributes with the defined tokens. An attribute is defined by a (user-defined) type, and a list of one or more token names (terminals) it is associated with. The following standard scanner types are pre-defined: *integer*, *real*, *boolean*, *char* and *string*. The type *string* is defined as a sequence of up to 255 characters. The type *boolean* is defined as an enumerated type containing the type values *true* and *false*. User-defined types are also enumerated types.

An attribute can be assigned a value using type values or the return value of scanner actions. The assignment is made after (one of) the associated terminal(s) is scanned.

When the parser needs to insert a terminal because of error recovery, the terminal's attribute(s) are set to a default value. The default value of an attribute depends on the type of the attribute. An attribute of type *integer* or *real* is set tot zero. Attributes of type *boolean* are set to *false*. Attributes of type *char* are set to the null character. Attributes of type *string* are set to the empty string. Attributes having a user-defined enumerated type are set to the first value that is enumerated.

Standard scanner actions

A scanner action always has one parameter of type *string*. The argument of a scanner action call does not have to be specified; the scanner generator will fill in the currently scanned string (i.e. representation) for you. The following standard actions are available in the scanner generator:

`get_repr: string`	returns the representation unchanged,
`convert_to_upper: string`	returns the uppercase version of the representation,
`convert_to_lower: string`	returns the lowercase version of the representation,
`delete_spaces: string`	returns the representation without any space characters,
`int_encode: integer`	returns an integer-typed value of the representation,
`real_encode: real`	returns a real-typed value of the representation,
`char_encode: char`	returns a char-typed value of the representation, excluding quotes,
`bool_encode: boolean`	returns a boolean-typed value of the representation, and
`string_encode: string`	returns a string-typed value of the representation, excluding quotes.

Meta-grammar of token definitions

A token can be defined as a regular expression or as a (list of) keyword and/or symbol (alternatives). If one or more attributes are associated with the defined token, then all the associated attributes must be assigned a value. In Figure 6 the meta grammar of a token definition is shown.

If a token should be skipped (like white space or comment) the reserved word **`SKIP`** can be specified. The scanner is then instructed to skip the current token and to continue by reading the next token.

If one or more keyword tokens are defined, an extra token should be defined to recognise all the keywords. This extra token definition should contain the reserved word **`VERIFY`**. The scanner is then instructed to look up the scanned string in the keyword table. If found, the appropriate keyword token is returned. If no keyword is found an error is reported and the verify token name is returned. It is also possible to specify a reserved action (not displayed in Figure 6) instead of **`VERIFY`**. The reserved action first converts the scanned string and the result is searched in the keyword table. If found, the appropriate keyword token is returned. If no keyword is found, the token name is returned (no error report). This can be used in case a token definition can match both a keyword and an identifier at the same time (for an example see the initial scanner specification of Lesson 5: Segments).

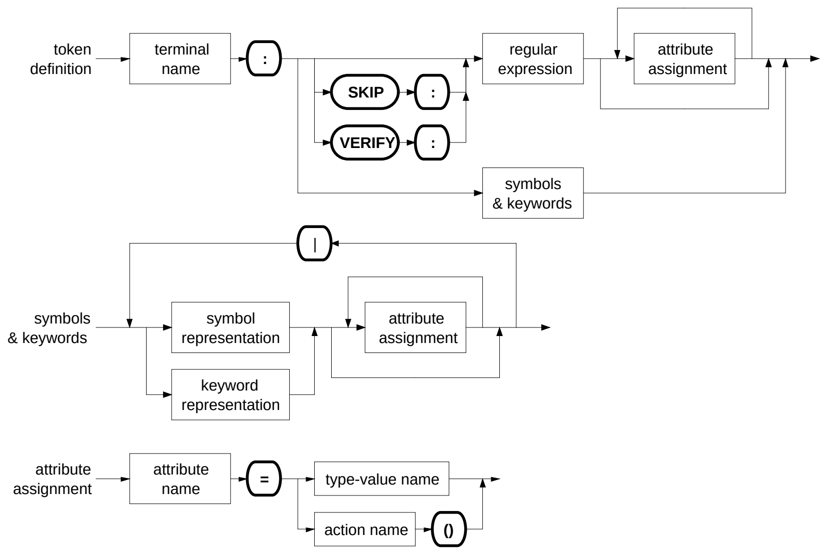

Figure 6: *Meta grammar of the token definition.*

A regular expression consists of pre-defined character sets, strings and syntactic operators. In Figure 7 the complete meta grammar is shown.

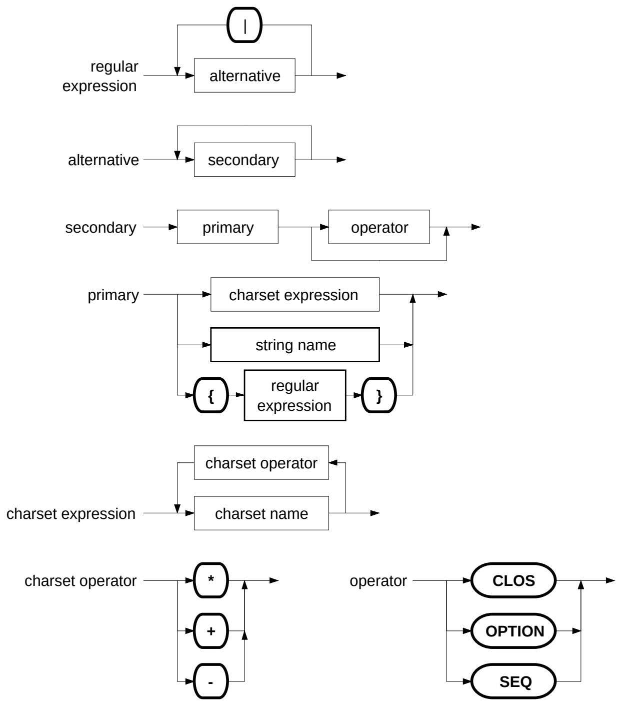

Figure 7: *Meta grammar of the regular expression.*

Scanner options

The following options can be specified:

NFA diagram	print a diagram of the non-deterministic finite automaton,
DFA diagram	print a diagram of the deterministic finite automaton,
*MDFA diagram**	print a diagram of the minimised deterministic finite automaton,
List names	print a list of the defined names,
List all names	print a list of the reserved and defined names,
Subset Trace	print the subsets of NFA states forming the DFA states,
Minimise trace	print the sets of DFA states being combined into one MDFA state, and
*Generate**	generate the scanner, if no errors occur.

The options marked with a * are default active. All output is written into the *.lst* file.