Regular expressions

In Pascal, an identifier is a letter followed by zero or more letters or digits, that is, and identifier is a member of the set defined by $L(L \cup D)^*$.

The notation we introduced so far is comfortable for mathematics but not for computers. Let us introduce another notation, called **regular expressions**, for describing the same languages and define its meaning in terms of the mathematical notation.

With this notation, we might define Pascal identifiers as

$$\textbf{letter}\ (\textbf{letter} \mid \textbf{digit})^\star$$

where the vertical bar means "or", the parentheses group subexpressions, the star means "zero or more instances of" the previous expression and juxtaposition means concatenation.

Regular expressions (continued)

A regular expression r is built up out of simpler regular expressions using a set of rules, as follows. Let Σ be an alphabet and $L(r)$ the language denoted by r.

1. ϵ is a regular expression that denotes $\{\varepsilon\}$.
2. If $a \in \Sigma$, then a is a regular expression that denotes $\{a\}$. This is ambiguous: a can denote a language, a word or a letter — it depends on the context.
3. Assume r and s denote the languages $L(r)$ and $L(s)$; a denotes a letter. Then
   3.1 $r \mid s$ is a regular expression denoting $L(r) \cup L(s)$.
   3.2 rs is a regular expression denoting $L(r)L(s)$.
   3.3 $r^\star$ is a regular expression denoting $(L(r))^*$.
   3.4 $\overline{a}$ is a regular expression denoting $\Sigma \backslash \{a\}$.

Regular expressions (continued)

A language described by a regular expression is a **regular language**.

Rules 1 and 2 form the base of the definition. Rule 3 provides the inductive step.

Unnecessary parentheses can be avoided in regular expressions if

- the unary operator * has the highest precedence and is left associative,
- concatenation has the second highest precedence and is left associative,
- $|$ has the lowest precedence and is left associative.

Under those conventions, $(a) \mid ((b)^\star(c))$ is equivalent to $a \mid b^\star c$.

Both expressions denote the language containing either the string a or zero or more b's followed by one c: $\{a, c, bc, bbc, bbbc, \ldots\}$.

Regular expressions/Examples

- The regular expression $a \mid b$ denotes the set $\{a, b\}$.
- The regular expression $(a \mid b)(a \mid b)$ denotes $\{aa, ab, ba, bb\}$, the set of all strings of a's and b's of length two. Another regular expression for the set is $aa \mid ab \mid ba \mid bb$.
- The regular expression $a^\star$ denotes the set of all strings of zero or more a's, i.e. $\{\varepsilon, a, aa, aaa, \dots\}$.
- The regular expression $(a \mid b)^\star$ denotes the set of all strings containing zero of more instances of an a or b, that is the language of all words made of a's and b's. Another expression is $(a^\star b^\star)^\star$.

Regular expressions/Algebraic laws

If two regular expressions r and s denote the same language, we say r and s are **equivalent** and write $r = s$.

Law	Description
$r \mid s = s \mid r$	$\mid$ is commutative
$r \mid (s \mid t) = (r \mid s) \mid t$	$\mid$ is associative
$(rs)t = r(st)$	concatenation is associative
$r(s \mid t) = rs \mid rt$ $(s \mid t)r = sr \mid tr$	concatenation distributes over $\mid$
$\epsilon r = r$ $r\epsilon = r$	ϵ is the identity element for the concatenation

Regular expressions/Algebraic laws (cont)

Law	Description
$r^{\star\star} = r^{\star}$	Kleene closure is idempotent
$r^{\star} = r^{+} \mid \epsilon$ $r^{+} = rr^{\star}$	Kleene closure and positive closure are closely linked

Regular definitions

It is convenient to give names to regular expressions and define new regular expressions using these names as if they were symbols.

If Σ is an alphabet, then a **regular definition** is a series of definitions of the form

$$\begin{aligned} d_1 &\rightarrow r_1 \\ d_2 &\rightarrow r_2 \\ &\dots \\ d_n &\rightarrow r_n \end{aligned}$$

where each d_i is a distinct name and each r_i is a regular expression over the alphabet $\Sigma \cup \{d_1, d_2, \dots, d_{i-1}\}$, i.e. the basic symbols and the previously defined names. The restriction to d_j such $j < i$ allows to construct a regular expression over Σ only by repeatedly replacing all the names in it.

Regular definitions/Examples

As we have stated, the set of Pascal identifiers can be defined by the regular definitions

$$
\begin{aligned}
\textbf{letter} &\rightarrow \mathtt{A} \mid \mathtt{B} \mid \ldots \mid \mathtt{Z} \mid \mathtt{a} \mid \mathtt{b} \mid \ldots \mid \mathtt{z} \\
\textbf{digit} &\rightarrow \mathtt{0} \mid \mathtt{1} \mid \mathtt{2} \mid \mathtt{3} \mid \mathtt{4} \mid \mathtt{5} \mid \mathtt{6} \mid \mathtt{7} \mid \mathtt{8} \mid \mathtt{9} \\
\textbf{id} &\rightarrow \textbf{letter}\ (\textbf{letter} \mid \textbf{digit})^{\star}
\end{aligned}
$$

Unsigned numbers in Pascal are strings like `5280`, `39.37`, `6.336E4` or `1.894E-4`.

$$
\begin{aligned}
\textbf{digit} &\rightarrow \mathtt{0} \mid \mathtt{1} \mid \mathtt{2} \mid \mathtt{3} \mid \mathtt{4} \mid \mathtt{5} \mid \mathtt{6} \mid \mathtt{7} \mid \mathtt{8} \mid \mathtt{9} \\
\textbf{digits} &\rightarrow \textbf{digit}\ \textbf{digit}^{\star} \\
\textbf{optional\_fraction} &\rightarrow \mathtt{.}\ \textbf{digits} \mid \epsilon \\
\textbf{optional\_exponent} &\rightarrow (\mathtt{E}\ (\mathtt{+} \mid \mathtt{-} \mid \epsilon)\ \textbf{digits}) \mid \epsilon \\
\textbf{num} &\rightarrow \textbf{digits}\ \textbf{optional\_fraction}\ \textbf{optional\_exponent}
\end{aligned}
$$

Regular definitions/Shorthands

Certain constructs occur so frequently in regular expressions that it is convenient to introduce notational shorthands for them.

Zero or one instance. The unary operator ? means "zero or one instance of." Formally, by definition, if r is a regular expression then $r? = r \mid \epsilon$. In other words, $(r)?$ denotes the language $L(r) \cup \{\varepsilon\}$.

$$
\begin{aligned}
\textbf{digit} &\rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
\textbf{digits} &\rightarrow \textbf{digit}^{+} \\
\textbf{optional\_fraction} &\rightarrow (\texttt{.}\ \textbf{digits})? \\
\textbf{optional\_exponent} &\rightarrow (\texttt{E}\ (\texttt{+} \mid \texttt{-})?\ \textbf{digits})? \\
\textbf{num} &\rightarrow \textbf{digits optional\_fraction optional\_exponent}
\end{aligned}
$$

Regular definitions/Shorthands (cont)

It is also possible to write:

$$
\begin{aligned}
\textbf{digit} &\rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
\textbf{digits} &\rightarrow \textbf{digit}^{+} \\
\textbf{fraction} &\rightarrow \texttt{.}\ \textbf{digits} \\
\textbf{exponent} &\rightarrow \texttt{E}\ (\texttt{+} \mid \texttt{-})?\ \textbf{digits} \\
\textbf{num} &\rightarrow \textbf{digits}\ \textbf{fraction}?\ \textbf{exponent}?
\end{aligned}
$$

Regular definitions/Shorthands (cont)

If we want to specify the characters `?`, `*`, `+`, `|`, we write them with a preceding backslash, e.g. `\?`, or between double-quotes, e.g. `"?"`. Then, of course, the character double-quote must have a backslash: `\"`

It is also sometimes useful to match against end of lines and end of files: `\n` stands for the control character “end of line” and **$** is for “end of file”.

Non-regular languages

Some languages cannot be described by any regular expression.

For example, the language of balanced parentheses cannot be recognised by any regular expression: (), (()), ()(), ((())()) etc.

Another example is the C programming language: it is not a regular language because it contains embedded blocs between `{` and `}`. Therefore, a lexer cannot recognise valid C programs: we need a parser.