Front End: Lexical Analysis

The Structure of a Compiler

Constructing a Lexical Analyser

- By hand: Identify *lexemes* in input and return *tokens*
- Automatically: *Lexical-Analyser generator*
- We will learn about *Lex*
- First we need to introduce:
  - Regular expressions
  - Non-deterministic automata
  - Deterministic automata

Scanning and Parsing

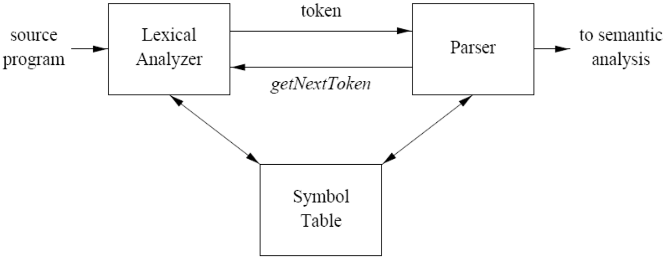


Figure 3.1: Interactions between the lexical analyzer and the parser

Important Notions

- Token: pair consisting of (`token-name`, `opt-value`)
- Pattern: form of the lexemes for a token
- Lexemes: sequence of characters matching the pattern for a token

Example

```
printf("Total = %d/n", score);
```

`printf` and `score` are lexemes for token **id** that matches pattern in Table 3.2

Classes of tokens

Token	Informal Description	Sample Lexemes
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	`<` or `>` or `<=` or `>=` or `==` or `!=`	`<=`, `!=`
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but `"`, surrounded by `"`'s	`"core dumped"`

Figure 3.2: Examples of tokens

Languages

- An alphabet is any finite set of symbols
- A string over an alphabet is a finite sequence of symbols from that alphabet
- A language is any countable set of strings over a fixed alphabet.

$|s|$ denotes the length of a string
ε is the string of length 0
Exponentiation:

$$\begin{array}{rcl} s^0 & = & \varepsilon \\ s^i & = & s^{i-1} \cdot s \end{array}$$

Operations on Strings

Parts of a string: example string "necessary"

- **prefix** : deleting zero or more tailing characters; eg: "nece"
- **suffix** : deleting zero or more leading characters; eg: "ssary"
- **substring** : deleting prefix and suffix; eg: "ssa"
- **subsequence** : deleting zero or more not necessarily contiguous symbols; eg: "ncsay"
- **proper** prefix, suffix, substring or subsequence: one that cannot equal to the original string;

Operations on Languages

Operation	Definition and Notation
Union of L and M	$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$
Concatenation of L and M	$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$
Kleene closure of L	$L^* = \cup_{i=0}^{\infty} L^i$
Positive closure of L	$L^+ = \cup_{i=1}^{\infty} L^i$

Figure 3.6: Definitions of operations on languages

We define:

$$L^0 = \{\varepsilon\}$$
$$L^i = L^{i-1}L.$$

Regular Expressions

Definition

A regular expression is defined inductively as follows:

- Basis
  - ε is a regular expression denoting the language $L(\varepsilon) = \{\varepsilon\}$
  - If $a \in \Sigma$ then **a** is a regular expression denoting $L(\mathbf{a}) = \{a\}$
- Induction: if r and s are regular expression with languages $L(\mathbf{r})$ and $L(\mathbf{s})$
  - $(r)|(s)$, $(r)(s)$, $(r)^*$, (r) are r.e. denoting $L(\mathbf{r}) \cup L(\mathbf{s})$, $L(\mathbf{r})L(\mathbf{s})$, $(L(\mathbf{r}))^*$ and $L(\mathbf{r})$.

Examples and Properties

Let $\Sigma = \{a, b\}$.

- $a|b$ denotes the language $\{a, b\}$
- What are the laguages denoted by $(a|b)(a|b)$, a^*, $(a|b)^*$, $a|a^*b$.

LAW	DESCRIPTION
$r\|s = s\|r$	$\|$ is commutative
$r\|(s\|t) = (r\|s)\|t$	$\|$ is associative
$r(st) = (rs)t$	Concatenation is associative
$r(s\|t) = rs\|rt;\ (s\|t)r = sr\|tr$	Concatenation distributes over $\|$
$\epsilon r = r\epsilon = r$	ϵ is the identity for concatenation
$r^* = (r\|\epsilon)^*$	ϵ is guaranteed in a closure
$r^{**} = r^*$	* is idempotent

Figure 3.7: Algebraic laws for regular expressions

Non-regular sets

- **Balanced or nested construct**
  - **Example:**
    if $cond_1$ **then if** $cond_2$ **then** $\cdots$ **else** $\cdots$ **else** $\cdots$
  - **Can be recognized by context free grammars.**
- **Matching strings:**
  - $\{wcw\}$, **where** w **is a string of** a**'s and** b**'s and** c **is a legal symbol.**
  - **Cannot be recognized even using context free grammars.**
- **Remark: anything that needs to "memorize" "non-constant" amount of information happened in the past cannot be recognized by regular expressions.**

Recognition of Tokens

Problem: Find prefixes of the input string that match the patterns.

Example

Consider the following grammar

$$
\begin{array}{lcl}
\mathit{stmt} & \rightarrow & \textbf{if } \mathit{expr} \textbf{ then } \mathit{stmt} \\
 & | & \textbf{if } \mathit{expr} \textbf{ then } \mathit{stmt} \textbf{ else } \mathit{stmt} \\
 & | & \epsilon \\
\mathit{expr} & \rightarrow & \mathit{term} \textbf{ relop } \mathit{term} \\
 & | & \mathit{term} \\
\mathit{term} & \rightarrow & \textbf{id} \\
 & | & \textbf{number}
\end{array}
$$

Figure 3.10: A grammar for branching statements

Example (ctdn.)

$$
\begin{array}{rcl}
digit & \rightarrow & \texttt{[0-9]} \\
digits & \rightarrow & digit^{+} \\
number & \rightarrow & digits\ (\texttt{.}\ digits)?\ (\ \texttt{E}\ \texttt{[+-]}?\ digits\ )? \\
letter & \rightarrow & \texttt{[A-Za-z]} \\
id & \rightarrow & letter\ (\ letter \mid digit\ )^{*} \\
if & \rightarrow & \texttt{if} \\
then & \rightarrow & \texttt{then} \\
else & \rightarrow & \texttt{else} \\
relop & \rightarrow & \texttt{<} \mid \texttt{>} \mid \texttt{<=} \mid \texttt{>=} \mid \texttt{=} \mid \texttt{<>}
\end{array}
$$

Figure 3.11: Patterns for tokens of Example 3.8

Example (ctdn.)

LEXEMES	TOKEN NAME	ATTRIBUTE VALUE
Any *ws*	–	–
`if`	**if**	–
`then`	**then**	–
`else`	**else**	–
Any *id*	**id**	Pointer to table entry
Any *number*	**number**	Pointer to table entry
`<`	**relop**	`LT`
`<=`	**relop**	`LE`
`=`	**relop**	`EQ`
`<>`	**relop**	`NE`
`>`	**relop**	`GT`
`>=`	**relop**	`GE`

Figure 3.12: Tokens, their patterns, and attribute values

Language Recognisers

Definition

A Finite Automata is a transition graph that *recognises* whether an input string belongs to a given *regular language* or not.

- Nondeterministic Finite Automata (NFA)
- Deterministic Finite Automata (DFA)

Both recognise the same languages, i.e. the regular languages.

NFA

A NFA consists of:

- A finite set of states S
- A alphabet Σ of input symbols
- A transition function $\texttt{move} : S \times \Sigma \cup \{\varepsilon\} \to \mathcal{P}(S)$
- A initial state $s_0 \in S$
- A set of accepting states $F \subseteq S$.

An Example of NFA

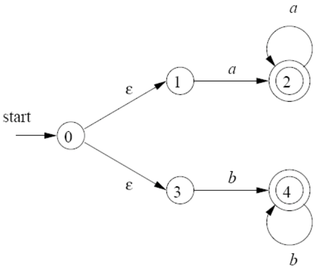
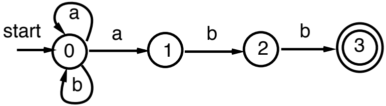


Figure 3.26: NFA accepting $\mathbf{aa}^*|\mathbf{bb}^*$

Executing a NFA

An NFA accepts an input string x if and only if there is some path in the transition graph initiating from the starting state to some accepting state such that the edge labels along the path spell out x.

(a|b)*abb input string: **aabb**



	a	b
0	**{0,1}**	**{0}**
1		**{2}**
2		**{3}**

$0 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 3$ **Accept!**

$0 \xrightarrow{a} 0 \xrightarrow{a} 0 \xrightarrow{b} 0 \xrightarrow{b} 0$ this is not an accepting path

DFA

A DFA is a special case of NFA where

- there are no ε-transitions
- For each $s \in S$ and $a \in \Sigma$ there is exactly one transition from s labelled a.

How to execute a DFA?

An Example of DFA

```
s = s0;
c = nextChar();
while ( c != eof ) {
        s = move(s, c);
        c = nextChar();
}
if ( s is in F ) return "yes";
else return "no";
```

Figure 3.27: Simulating a DFA

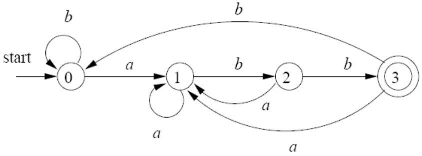


Figure 3.28: DFA accepting $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

How to implement a NFA?

Recall: A NFA for language $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$ is

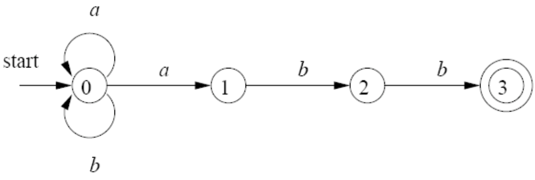


Figure 3.24: A nondeterministic finite automaton

Because of the non-deterministic choices, simulating a NFA is not as straightforward as for a DFA.

Convert NFA in DFA!

Algorithm for *Subset Construction*

Given NFA *N* constructs DFA *D* by simulating in parallel all the moves *N* can make on a input string.

OPERATION	DESCRIPTION
$\epsilon\text{-}closure(s)$	Set of NFA states reachable from NFA state s on ϵ-transitions alone.
$\epsilon\text{-}closure(T)$	Set of NFA states reachable from some NFA state s in set T on ϵ-transitions alone; $= \cup_{s \text{ in } T}\ \epsilon\text{-}closure(s)$.
$move(T, a)$	Set of NFA states to which there is a transition on input symbol a from some state s in T.

Figure 3.31: Operations on NFA states

Example

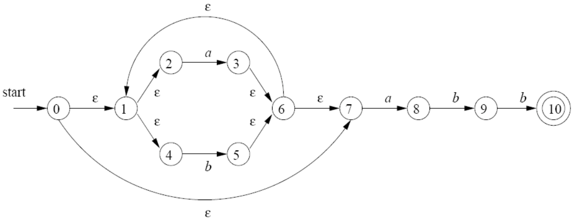


Figure 3.34: NFA N for $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

NFA State	DFA State	a	b
$\{0, 1, 2, 4, 7\}$	A	B	C
$\{1, 2, 3, 4, 6, 7, 8\}$	B	B	D
$\{1, 2, 4, 5, 6, 7\}$	C	B	C
$\{1, 2, 4, 5, 6, 7, 9\}$	D	B	E
$\{1, 2, 4, 5, 6, 7, 10\}$	E	B	C

Figure 3.35: Transition table $Dtran$ for DFA D

Example (ctdn.)

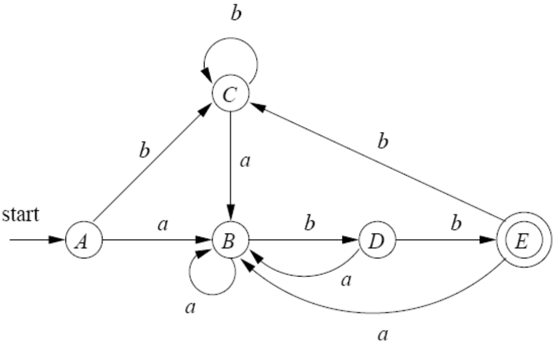


Figure 3.36: Result of applying the subset construction to Fig. 3.34