Lexical Analysis

Reinhard Wilhelm
Universität des Saarlandes
wilhelm@cs.uni-sb.de
and
Mooly Sagiv
Tel Aviv University
sagiv@math.tau.ac.il

2. November 2007
Subjects

- Role of lexical analysis
- Regular languages, regular expressions
- Finite automata
- From regular expressions to finite automata
- A language for specifying lexical analysis
- The generation of a scanner
- Flex
“Standard” Structure

source(text) → lexical analysis (7) → tokenized-program → syntax analysis (8) → syntax-tree → semantic-analysis (9) → decorated syntax-tree → optimizations (10) → intermediate rep.

finite automata → pushdown automata → attribute grammar evaluators → abstract interpretation + transformations...

Lexical Analysis
“Standard” Structure cont’d

intermediate rep. → code-generation(11, 12) → machine-program → tree automata + dynamic programming + ...
Lexical Analysis (Scanning)

- **Functionality**
  - **Input:** program as sequence of characters
  - **Output:** program as sequence of symbols (tokens)

- **Produce listing**

- **Report errors, symbols illegal in the programming language**

- **Screening subtask:**
  - Identify language keywords and standard identifiers
  - Eliminate “white-space”, e.g., consecutive blanks and newlines
  - Count line numbers
  - Construct table of all symbols occurring
Automatic Generation of Lexical Analyzers

- The symbols of programming languages can be specified by regular expressions.

- Examples:
  - program as a sequence of characters.
  - (alpha (alpha | digit)*) for Pascal identifiers
  - “(‘*‘ until ‘‘*‘‘)‘‘ for Pascal comments

- The recognition of input strings can be performed by a finite automaton.

- A table representation or a program for the automaton is automatically generated from a regular expression.
Automatic Generation of Lexical Analyzers cont’d
Notations

A language, \( L \), is a set of words, \( x \), over an alphabet, \( \Sigma \).

- \( a_1 a_2 \ldots a_n \) \quad A word over \( \Sigma \)
- \( a_i \in \Sigma \)
- \( \varepsilon \) \quad The empty word
- \( \Sigma^n \) \quad The words of length \( n \) over \( \Sigma \)
- \( \Sigma^* \) \quad The finite words over \( \Sigma \)
- \( \Sigma^+ \) \quad The non-empty finite words over \( \Sigma \)
- \( x.y \) \quad The concatenation of \( x \) and \( y \)

Language Operations

\[
\begin{align*}
L_1 \cup L_2 & \quad \text{Union} \\
L_1 L_2 & = \{x.y|x \in L_1, y \in L_2\} \quad \text{Concatenation} \\
\overline{L} & = \Sigma^* - L \quad \text{Complement} \\
L^n & = \{x_1 \ldots x_n|x_i \in L, 1 \leq i \leq n\} \\
L^* & = \bigcup_{n \geq 0} L^n \quad \text{Closure} \\
L^+ & = \bigcup_{n \geq 1} L^n
\end{align*}
\]
Regular Languages

Defined inductively

- \( \emptyset \) is a regular language over \( \Sigma \)
- \( \{ \varepsilon \} \) is a regular language over \( \Sigma \)
- For all \( a \in \Sigma \), \( \{ a \} \) is a regular language over \( \Sigma \)
- If \( R_1 \) and \( R_2 \) are regular languages over \( \Sigma \), then so are:
  - \( R_1 \cup R_2 \),
  - \( R_1 R_2 \), and
  - \( R_1^* \)
Regular Expressions and the Denoted Regular Languages

Defined inductively

- $\emptyset$ is a regular expression over $\Sigma$ denoting $\emptyset$,
- $\varepsilon$ is a regular expression over $\Sigma$ denoting $\{\varepsilon\}$,
- For all $a \in \Sigma$, $a$ is a regular expression over $\Sigma$ denoting $\{a\}$,
- If $r_1$ and $r_2$ are regular expressions over $\Sigma$ denoting $R_1$ and $R_2$, resp., then so are:
  - $(r_1|r_2)$, which denotes $R_1 \cup R_2$,
  - $(r_1r_2)^{-1}$, which denotes $R_1R_2$, and
  - $(r_1)^*$, which denotes $R_1^*$.
- Metacharacters, $\emptyset, \varepsilon, (, |,*$ don’t really exist, are replaced by their non-underlined versions.
  Attention: Clash between characters in $\Sigma$ and metacharacters $\{(, |,*\}$
## Example

<table>
<thead>
<tr>
<th>Expression</th>
<th>Language</th>
<th>Example Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ab$</td>
<td>${a, b}$</td>
<td>$a, b$</td>
</tr>
<tr>
<td>$ab^*a$</td>
<td>${a}{b}^*{a}$</td>
<td>$aa, aba, abba, abbba, \ldots$</td>
</tr>
<tr>
<td>$(ab)^*$</td>
<td>${ab}^*$</td>
<td>$\varepsilon, ab, abab, \ldots$</td>
</tr>
<tr>
<td>$abba$</td>
<td>${abba}$</td>
<td>$abba$</td>
</tr>
</tbody>
</table>
Regular Expressions for Symbols (Tokens)

Alphabet for the symbol classes listed below:

\[ \Sigma = \]

- integer-constant
- real-constant
- identifier
- string
- comments
- matching-parentheses?
Finite Automaton

Input Tape

Actual State

Control
A Non-Deterministic Finite Automaton (NFA)

\[ M = \langle \Sigma, Q, \Delta, q_0, F \rangle \] where:

- \( \Sigma \) — finite alphabet
- \( Q \) — finite set of states
- \( q_0 \in Q \) — initial state
- \( F \subseteq Q \) — final states
- \( \Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q \) — transition relation

May be represented as a transition diagram

- Nodes — States
- \( q_0 \) has a special “entry” mark
- final states doubly encircled
- An edge from \( p \) into \( q \) labeled by \( a \) if \( (p, a, q) \in \Delta \)
Example: Integer and Real Constants

<table>
<thead>
<tr>
<th></th>
<th>$D_i \in {0, 1, \ldots, 9}$</th>
<th>$E$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{1,2}</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
<tr>
<td>1</td>
<td>{1}</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
<tr>
<td>2</td>
<td>{2}</td>
<td>{3}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>3</td>
<td>{4}</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
<tr>
<td>4</td>
<td>{4}</td>
<td>\emptyset</td>
<td>{5}</td>
</tr>
<tr>
<td>5</td>
<td>{6}</td>
<td>\emptyset</td>
<td>{7}</td>
</tr>
<tr>
<td>6</td>
<td>{7}</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
<tr>
<td>7</td>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

$q_0 = 0$

$F = \{1, 7\}$
Maximal Munch strategy

Find longest prefix of remaining input that is a legal symbol.

- first input character of the scanner — first “non-consumed” character,
- final state and transition under the next character: make transition and remember position,
- final state and no transition under the next character: Symbol found,
- actual state not final and no transition under the next character: backtrack to last passed final state
  - There is none: Illegal string
  - Otherwise: Actual symbol ended there.

Warning: Certain overlapping symbol definitions will result in quadratic runtime: Example: \((a|a^*;\)
Other Example Automata

- integer-constant
- real-constant
- identifier
- string
- comments
The Language Accepted by an Automaton

- $M = \langle \Sigma, Q, \Delta, q_0, F \rangle$
- For $q \in Q$, $w \in \Sigma^*$, $(q, w)$ is a configuration
- The step binary relation on configurations is defined by:

\[(q, aw) \vdash_M (p, w)\]

if $(q, a, p) \in \Delta$
- The reflexive transitive closure of $\vdash_M$ is denoted by $\vdash^*_M$
- The language accepted by $M$

\[L(M) = \{ w \mid w \in \Sigma^* \mid \exists q_f \in F : (q_0, w) \vdash^*_M (q_f, \varepsilon) \}\]
From Regular Expressions to Finite Automata

Theorem

(i) For every regular language $R$, there exists an NFA $M$, such that $L(M) = R$.

(ii) For every regular expression $r$, there exists an NFA that accepts the regular language defined by $r$. 
A Constructive Proof for (ii) (Algorithm)

- A regular language is defined by a regular expression $r$
- Construct an “NFA” with one final state, $q_f$, and the transition $q_0 \xrightarrow{r} q_f$

- Decompose $r$ and develop the NFA according to the following rules

\[ q \xrightarrow{r_1 r_2} p \quad \Rightarrow \quad q \xrightarrow{r_1} q_1 \xrightarrow{r_2} p \]

\[ q \xrightarrow{r_1 r_2} p \quad \Rightarrow \quad q \xrightarrow{r_1} q_1 \xrightarrow{r_2} p \]

\[ q \xrightarrow{r^*} p \quad \Rightarrow \quad q \xrightarrow{\varepsilon} q_1 \xrightarrow{r} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{r} q_2 \xrightarrow{\varepsilon} p \]

until only transitions under single characters and $\varepsilon$ remain.
Examples

- $a(a|0)^*$ over $\Sigma = \{a, 0\}$

- Identifier

- String
Nondeterminism

- Several transitions may be possible under the same character in a given state
- $\varepsilon$-moves (next character is not read) may “compete” with non-$\varepsilon$-moves.
- Deterministic simulation requires “backtracking”
Deterministic Finite Automaton (DFA)

- No $\varepsilon$-transitions
- At most one transition from every state under a given character, i.e. for every $q \in Q, \ a \in \Sigma$,

$$|\{q' \mid (q, a, q') \in \Delta\}| \leq 1$$
From Non-Deterministic to Deterministic Automata

Theorem
For every NFA, $M = \langle \Sigma, Q, \Delta, q_0, F \rangle$ there exists a DFA, $M' = \langle \Sigma, Q', \delta, q_0', F' \rangle$ such that $L(M) = L(M')$.

A Scheme of a Constructive Proof (Algorithm)
Construct a DFA whose states are sets of states of the NFA. New state $S = \{q_1, \ldots, q_n\}$, if there is a word $w$ such that $(q_0, w) \vdash^*_M (q_i, \varepsilon)$ for exactly the $q_1, \ldots, q_n$
The Construction Algorithm

- Used in the construction: $\varepsilon$-$SS(q) = \{ p \mid (q, \varepsilon) \vdash_M^* (p, \varepsilon) \}$
- A DFA state in $Q'$ — a set of NFA original states $S \subseteq Q$
- Starts with $q'_0 = \varepsilon$-$SS(q_0)$ as the initial DFA state.
- Iteratively creates more states and more transitions.
- For every DFA state $S \subseteq Q$ and character $a \in \Sigma$,

$$
\delta(S, a) = \bigcup_{q \in S} \bigcup_{(q, a, p) \in \Delta} \varepsilon$-$SS(p)
$$

This may create a new state $\delta(S, a)$

- A DFA state $S$ is accepting (in $F'$) if there exists $q \in S$ such that $q \in F$
Example: $a(a|0)^*$
DFA minimization

DFA need not have minimal size, i.e. minimal number of states and transitions.  
$q$ and $p$ are undistinguishable iff for all words $w$ $(q,w) \vdash^*_M$ and $(p,w) \vdash^*_M$ lead into either $F'$ or $Q' - F'$. Undistinguishable states can be merged.
DFA minimization algorithm

- Input a DFA $M = \langle \Sigma, Q, \delta, q_0, F \rangle$
- Iteratively refine a partition of the set of states, where each set in the partition consists of states so far undistinguishable.
- Start with the partition $\Pi = \{F, Q - F\}$
- Refine the current $\Pi$ by splitting sets $S \in \Pi$ if there exist $q_1, q_2 \in S$ and $a \in \Sigma$ such that
  - $\delta(q_1, a) \in S_1$ and $\delta(q_2, a) \in S_2$ and $S_1 \neq S_2$
- Merge sets of undistinguishable states into a single state.
Example: $a(a|0)^*$
A Language for specifying lexical analyzers

\[(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^* \]
\[(\varepsilon).(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^* \]
\[(\varepsilon|E(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9))) \]
Descriptional Comfort

Character Classes:
Identical meaning for the DFA (exceptions!), e.g.,
\[ le = a \ - \ z \ A \ - \ Z \]
\[ di = 0 \ - \ 9 \]
Efficient implementation: Addressing the transitions indirectly through an array indexed by the character codes.

Symbol Classes:
Identical meaning for the parser, e.g.,
Identifiers
Comparison operators
Strings
Descriptional Comfort cont’d

Sequences of regular definitions:

\[ A_1 = R_1 \]
\[ A_2 = R_2 \]
\[ \ldots \]
\[ A_n = R_n \]
Sequences of Regular Definitions

Goal: Separate final states for each definition

1. Substitute right sides for left sides
2. Create an NFA for every regular expression separately;
3. Merge all the NFAs using $\varepsilon$ transitions from the start state;
4. Construct a DFA;
5. Minimize starting with partition

\[ \{F_1, F_2, \ldots, F_n, Q - \bigcup_{i=1}^{n} F_i\} \]
Lexical Analysis

Flex Specification

Definitions
%%

Rules
%%

C-Routines
%{
extern int line_number;
extern float atof(char *);
%}
DIG       [0-9]
LET       [a-zA-Z]
%
[=<>+-*]     { return(*yytext); }
({DIG}+)    { yylval.intc  = atoi(yytext); return(301); }
({DIG})*\.({DIG}+)(E(\+|-)?{DIG}+)?
    { yylval.realc = atof(yytext); return(302); }
"(\\.|[^\\\\])"   { strcpy(yylval.strc, yytext);
    return(303); }
"<="      { return(304); }
:=        { return(305); }
\./.      { return(306); }
Lexical Analysis

Flex Example cont’d

```
ARRAY { return(307); }
BOOLEAN { return(308); }
DECLARE { return(309); }
{LET}({LET}|{DIG})* { yylval.symb = look_up(yytext);
    return(310); }
[ \t]+ { /* White space */ }
\n { line_number++; }
. { fprintf(stderr,
    "WARNING: Symbol '%c' is illegal, ignored!\n", *yytext); }
```