Lexical Analysis

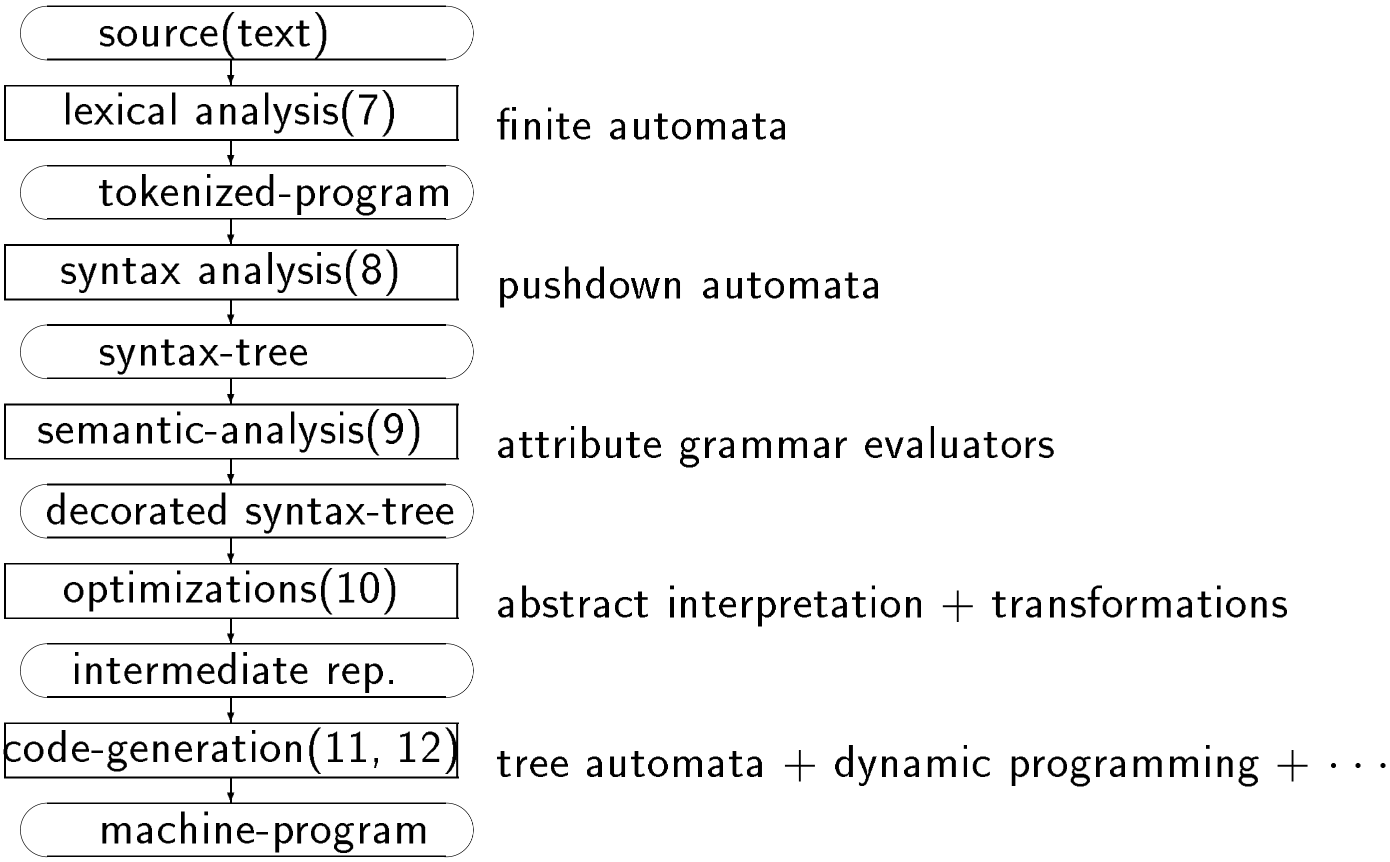
Mooly Sagiv
Tel Aviv University
sagiv@math.tau.ac.il
and
Reinhard Wilhelm
Universität des Saarlandes
wilhelm@cs.uni-sb.de

May 5, 1997

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” Compiler Structure



Lexical Analysis (Scanning)

- Functionality

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

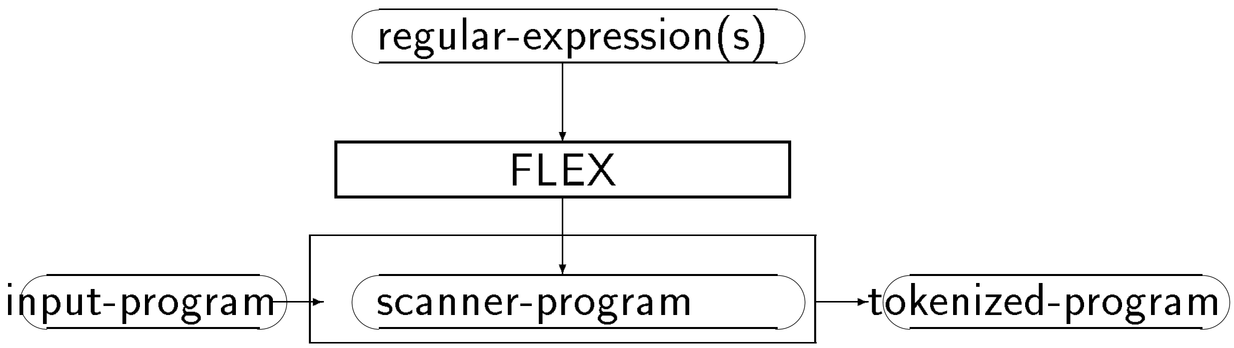
- Produce listing

- 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.



Notations

$x_1 x_2 \ldots x_n$, $x_i \in \Sigma$	A **word** over Σ
ε	The empty word
Σ^n	The words of length n over Σ
Σ^*	The finite words over Σ
Σ^+	The non-empty finite words over Σ
$x.y$	The concatenation of x and y

Language Operations

$L_1 \cup L_2$		Union
$L_1 L_2$	$\{x.y \mid x \in L_1, y \in L_2\}$	Concatenation
$\overline{L}$	$\Sigma^* - L$	Complement
L^n	$\{x_1 \ldots x_n \mid x_i \in L, 1 \leq i \leq n\}$	
L^*	$\bigcup_{n \geq 0} L^n$	Closure
L^+	$\bigcup_{n \geq 1} L^n$	

Regular Languages

Defined inductively

- ϕ is a **regular language over** Σ
- $\{\varepsilon\}$ is a **regular language over** Σ
- For all $a \in \Sigma$, $\{a\}$ is a **regular language over** Σ
- If R_1 and R_2 are regular languages over Σ then so are:
  - $R_1 \cup R_2$,
  - $R_1 R_2$, and
  - R_1^*

Regular Expressions and the Denoted Regular Languages

Defined inductively

- $\underline{\phi}$ is a **regular expression over** Σ denoting ϕ,
- $\underline{\epsilon}$ is a **regular expression over** Σ denoting $\{\varepsilon\}$,
- For all $a \in \Sigma$, a is a **regular expression over** Σ denoting $\{a\}$,
- If r_1 and r_2 are regular expressions over Σ denoting R_1 and R_2, resp., then so are:
  - $\underline{(}r_1\underline{|}r_2\underline{)}$ which denotes $R_1 \cup R_2$,
  - $\underline{(}r_1 r_2\underline{)}$ which denotes $R_1 R_2$, and
  - $\underline{(}r_1\underline{)}\underline{^*}$ which denotes R_1^*.
- Attention: Clash between characters in Σ and metacharacters $\{\underline{(}, \underline{)}, \underline{|}, \underline{^*}\}$

Example

Expression	Language	Example Words
$a \mid b$	$\{a, b\}$	a, b
ab^*a	$\{a\}\{b\}^*\{a\}$	$aa, aba, abba, abbba, \ldots$
$(ab)^*$	$\{ab\}^*$	$\varepsilon, ab, abab, \ldots$
$abba$	$\{abba\}$	$abba$

Regular Expressions for Symbols (Tokens)

integer-constant

real-constant

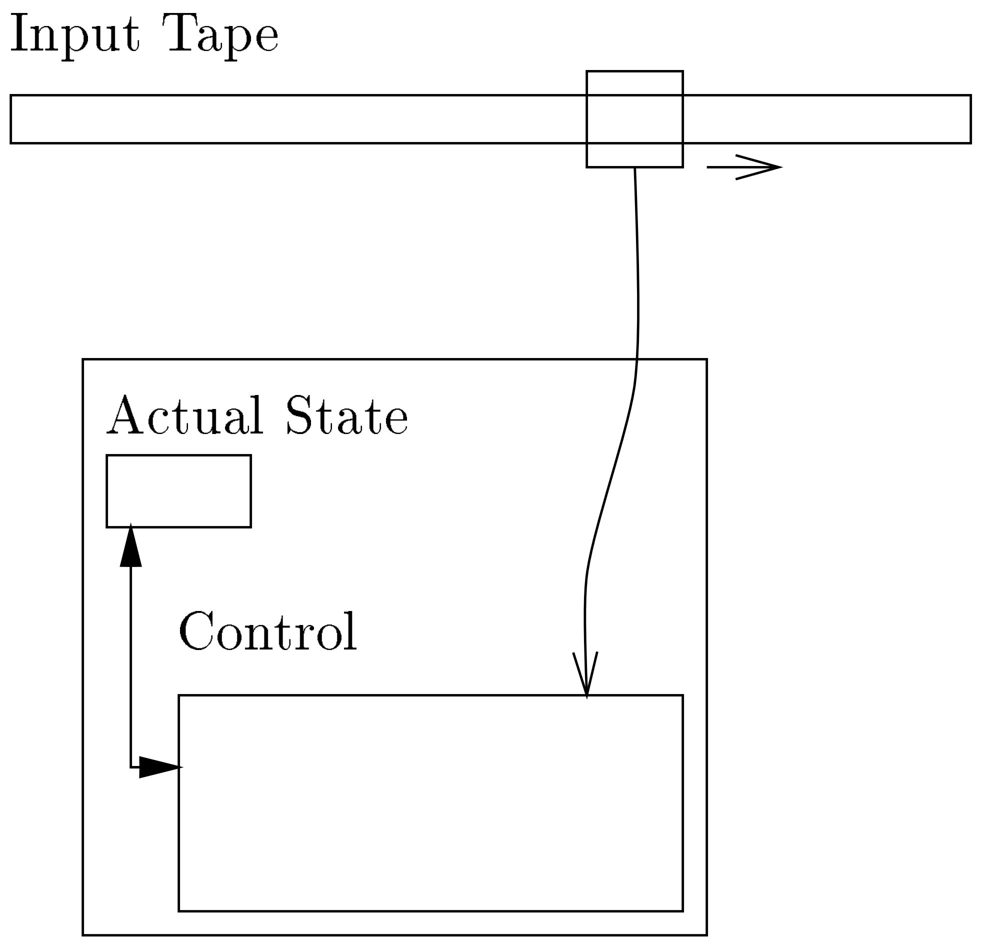
identifier

string

comments

matching-parentheses?

Finite Automaton



A Non-Deterministic Finite Automaton (NFA)

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

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

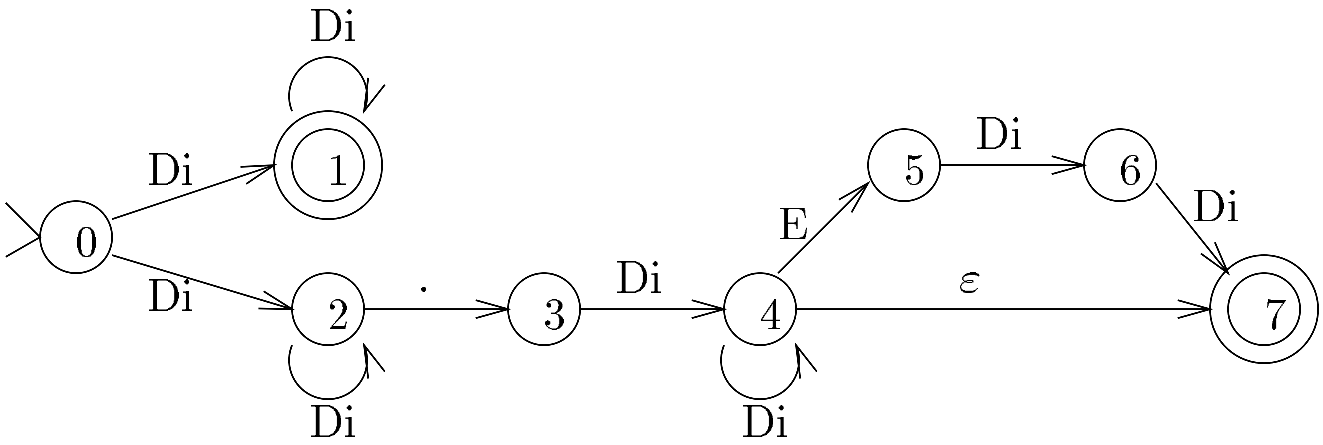
May be represented as a **transition diagram**

- 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

	0,1,. . . , 9	.	E	ε
0	{1,2}	∅	∅	∅
1	{1}	∅	∅	∅
2	{2}	{3}	∅	∅
3	{4}	∅	∅	∅
4	{4}	∅	{5}	{7}
5	{6}	∅	∅	∅
6	{7}	∅	∅	∅
7	∅	∅	∅	∅

$$q_0 = 0$$
$$F = \{1, 7\}$$



Conventions

Maximal Munch strategy:
find longest prefix of remaining input which is a legal symbol.

- The first input character of the scanner — the first “non-consumed” character
- 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
$$L(M) = \{w \mid w \in \Sigma^* \mid \exists q_f \in F : (q_0, w) \vdash_M^* (q_f, \epsilon)\}$$

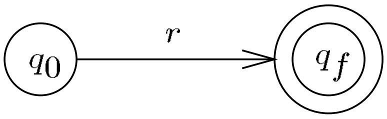
From Regular Expressions to Finite Automata

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

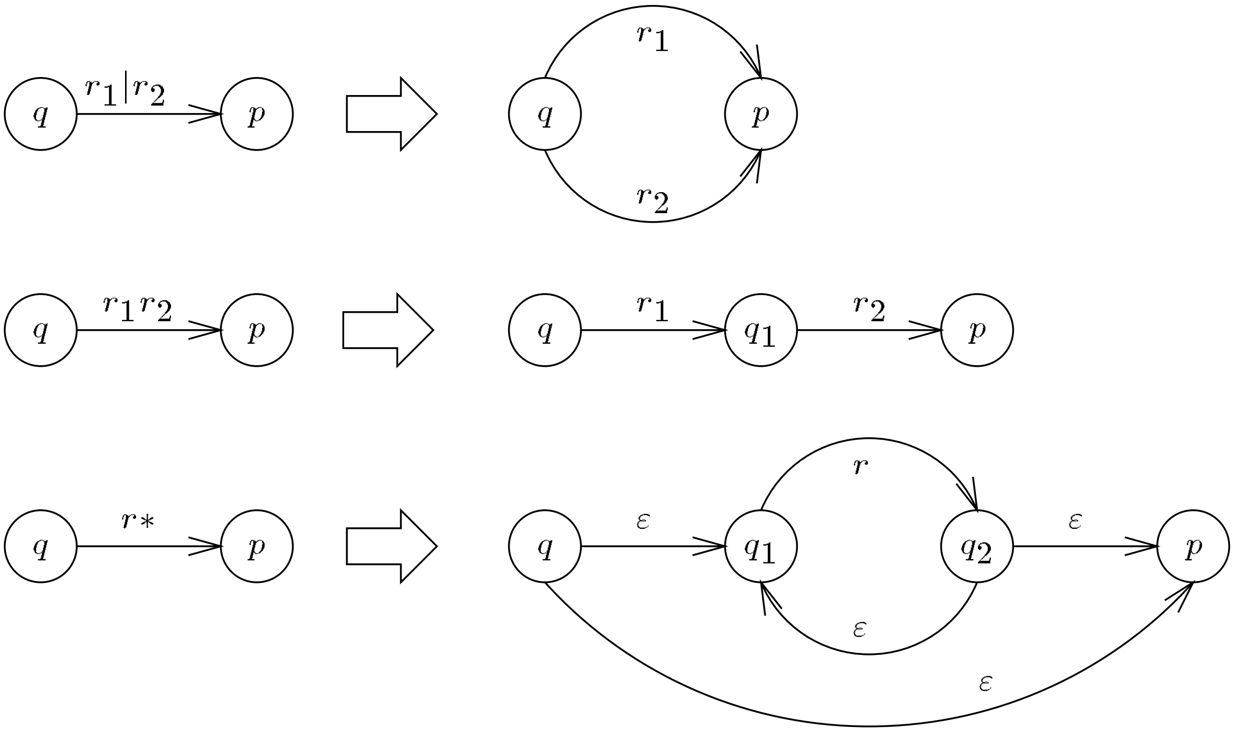
For every regular expression r, there exists an NFA, which accepts the regular language defined by r.

A Constructive Proof (Algorithm)

- A regular language is defined by a regular expression r
- Construct an "NFA" with one final state, q_f, and the transition



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



until only transitions under single characters and ε remain.

Examples

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

- Identifier

- String

Nondeterminism

- Many transitions may be possible under the same character in a given state
- ϵ-moves (next character is not read) may "compete" with non-ε-moves.
- Deterministic simulation requires "backtracking"

Deterministic Finite Automaton (DFA)

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

$$|\{q' \mid (q, a, q') \in \Delta\}| \leq 1$$

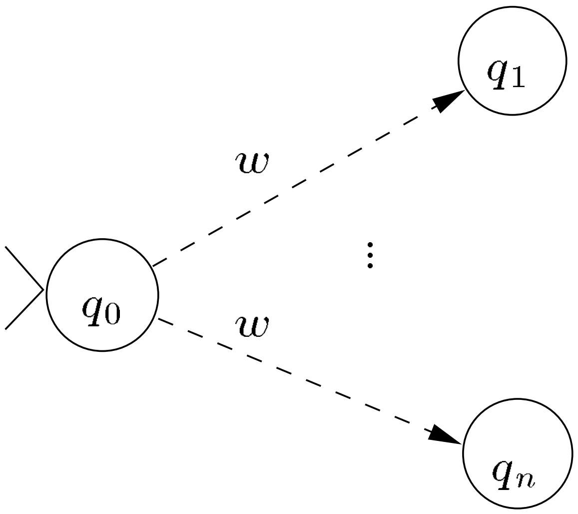
From Non-Deterministic to Deterministic Automata

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.

$S = \{q_1, \ldots, q_n\}$, 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:

$$\epsilon\text{-}SS(q) = \{p \mid (q, \epsilon) \vdash_M^* (p, \epsilon)\}$$

- A DFA state in Q' — a set of NFA original states $S \subseteq Q$

- Starts with $q_0' = \epsilon\text{-}SS(q_0)$ as the initial DFA state.

- Iteratively creates more states and more transitions.

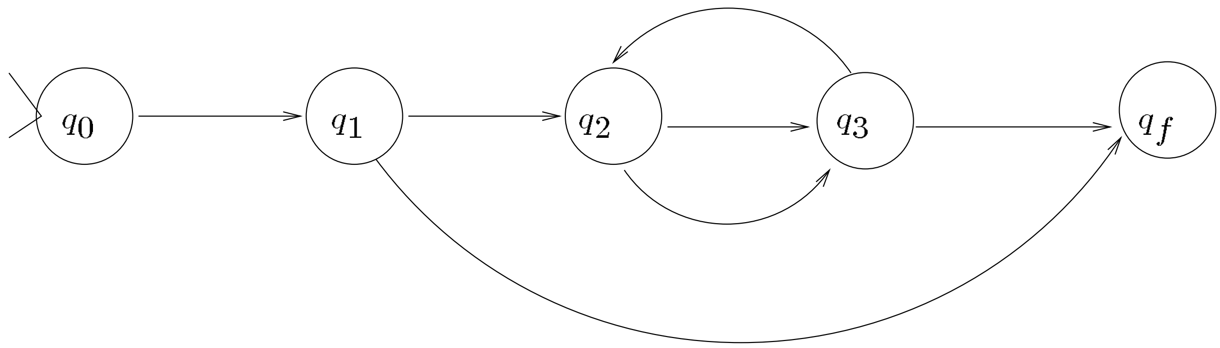
- For every DFA state $S \subseteq Q$ and character $a \in \Sigma$,

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

  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 $q \in F$

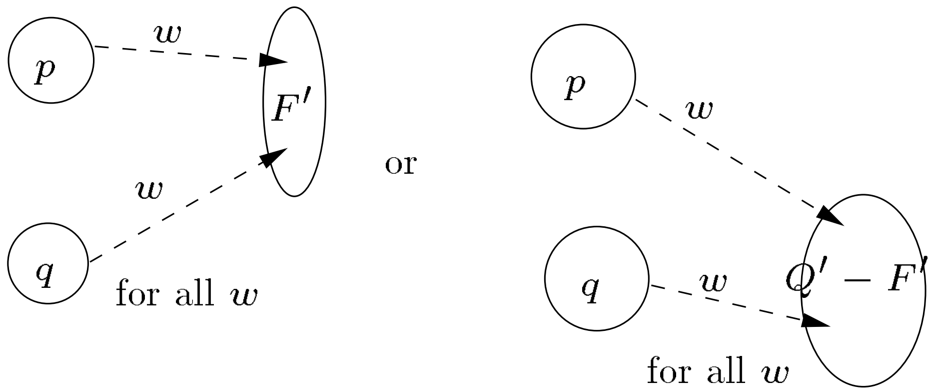
Example: $a(a|0)^*$



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 Π by splitting sets $S \in \Pi$ if there exist $q_1, q_2 \in S$ and $a \in \Sigma$ such that
  - $\delta(q_1, a) \in S_1$
  - $\delta(q_2, a) \in S_2$
  - $S_1 \neq S_2$
- Merge sets of undistinguishable states into a single state.

Example: $a(a|0)^*$

A Language for specifying lexical analyzers

$$
\begin{array}{l}
\quad (0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^* \\
(\varepsilon|.(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^* \\
(\varepsilon|E(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)))
\end{array}
$$

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

Sequences of regular definitions:

$$\begin{array}{lcl} A_1 & = & R_1 \\ A_2 & = & R_2 \\ & \ldots & \\ A_n & = & R_n \end{array}$$

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 ϵ transitions from the start state;
4. Construct a DFA;
5. Minimize starting with partition

$$\{F_1, F_2, \ldots, F_n, Q - \bigcup_{i=1}^{n} F_i\}$$

Flex Specification

Definitions
%%
Rules
%%
C-Routines

Flex Example

```
%{
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); }
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);}
%%
```