Lexical Analysis
(Scanning)

Lexical Analysis (Scanning)

Translates a stream of characters to a stream of tokens

```
f o o ␣ = ␣ a + ␣ bar(2, ␣ q);
```

ID EQUALS ID PLUS ID LPAREN NUM COMMA ID LPAREN SEMI

Token	Lexemes	Pattern
EQUALS	`=`	an equals sign
PLUS	`+`	a plus sign
ID	`a foo bar`	letter followed by letters or digits
NUM	`0 42`	one or more digits

Lexical Analysis

Goal: simplify the job of the parser.

Scanners are usually much faster than parsers.

Discard as many irrelevant details as possible (e.g., whitespace, comments).

Parser does not care that the the identifer is “supercalifragilisticexpialidocious.”

Parser rules are only concerned with tokens.

Describing Tokens

Alphabet: A finite set of symbols

Examples: $\{$ 0, 1 $\}$, $\{$ A, B, C, $\ldots$, Z $\}$, ASCII, Unicode

String: A finite sequence of symbols from an alphabet

Examples: ϵ (the empty string), Stephen, $\alpha\beta\gamma$

Language: A set of strings over an alphabet

Examples: $\emptyset$ (the empty language), $\{$ 1, 11, 111, 1111 $\}$, all English words, strings that start with a letter followed by any sequence of letters and digits

Operations on Languages

Let $L = \{$ ϵ, wo $\}$, $M = \{$ man, men $\}$

Concatenation: Strings from one followed by the other

$LM = \{$ man, men, woman, women $\}$

Union: All strings from each language

$L \cup M = \{\epsilon$, wo, man, men $\}$

Kleene Closure: Zero or more concatenations

$M^* = \{\epsilon, M, MM, MMM, \ldots\} =$
$\{\epsilon$, man, men, manman, manmen, menman, menmen, manmanman, manmanmen, manmenman, $\ldots\}$

Regular Expressions over an Alphabet Σ

A standard way to express languages for tokens.

1. ϵ is a regular expression that denotes $\{\epsilon\}$
2. If $a \in \Sigma$, a is an RE that denotes $\{a\}$
3. If r and s denote languages $L(r)$ and $L(s)$,
    - $(r)|(s)$ denotes $L(r) \cup L(s)$
    - $(r)(s)$ denotes $\{tu : t \in L(r), u \in L(s)\}$
    - $(r)^*$ denotes $\cup_{i=0}^{\infty} L^i$ ($L^0 = \{\epsilon\}$ and $L^i = LL^{i-1}$)

Regular Expression Examples

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

RE	Language
$a\|b$	$\{a, b\}$
$(a\|b)(a\|b)$	$\{aa, ab, ba, bb\}$
a^*	$\{\epsilon, a, aa, aaa, aaaa, \ldots\}$
$(a\|b)^*$	$\{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, \ldots\}$
$a\|a^*b$	$\{a, b, ab, aab, aaab, aaaab, \ldots\}$

Specifying Tokens with REs

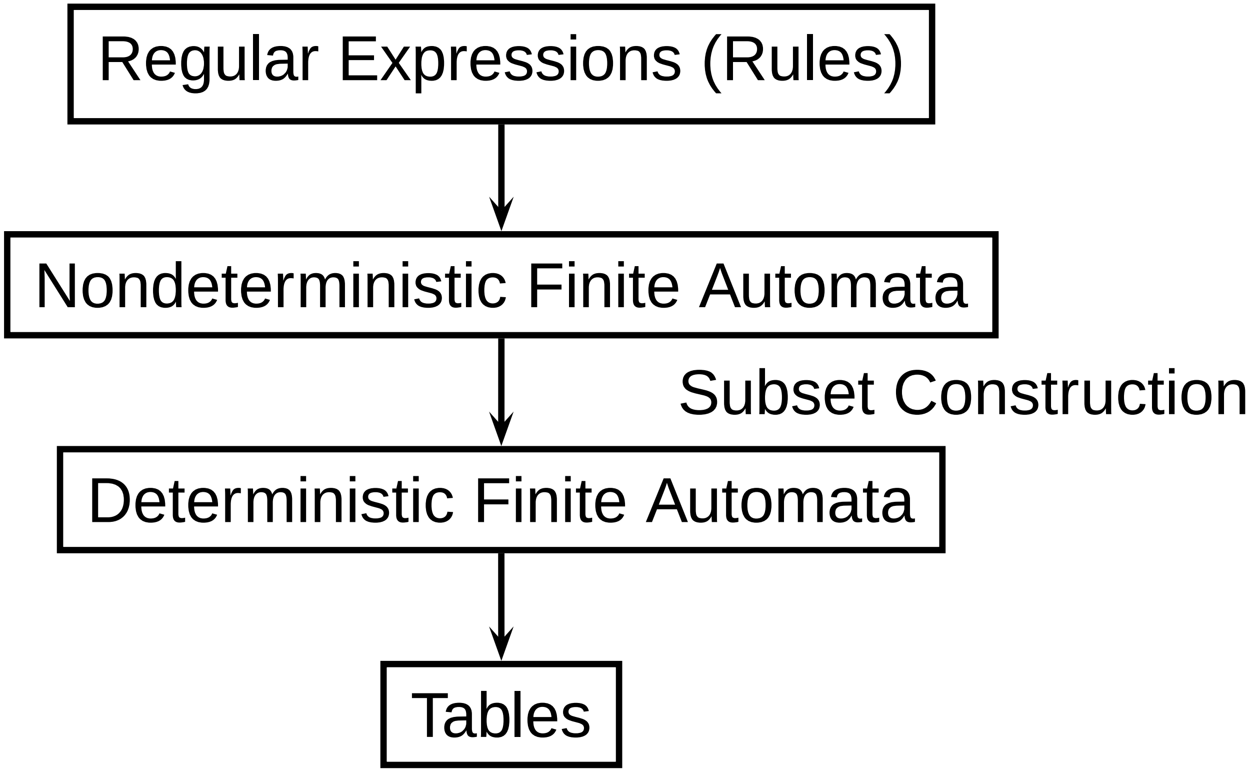
Typical choice: $\Sigma =$ ASCII characters, i.e.,
$\{␣, !, "$, $\#, \$, \ldots, 0, 1, \ldots, 9, \ldots, A, \ldots, Z, \ldots, \tilde{}\}$

letters: $A|B|\cdots|Z|a|\cdots|z$

digits: $0|1|\cdots|9$

identifier: **letter** $($ **letter** $|$ **digit** $)^*$

Implementing Scanners Automatically



Regular Expressions (Rules)

Nondeterministic Finite Automata

Subset Construction

Deterministic Finite Automata

Tables

The ANTLR Compiler Generator

Language and compiler for writing compilers

Running ANTLR on an ANTLR file produces Java source files that can be compiled and run.

ANTLR can generate

- Scanners (lexical analyzers)
- Parsers
- Tree walkers

An ANTLR File for a Simple Scanner

```
class CalcLexer extends Lexer;

LPAREN : ’(’ ;       // Rules for puctuation
RPAREN : ’)’ ;
STAR : ’*’ ;
PLUS : ’+’ ;
SEMI : ’;’ ;
protected            // Can only be used as a sub-rule
DIGIT : ’0’..’9’ ;   // Any character between 0 and 9
INT : (DIGIT)+ ;     // One or more digits

WS : (’ ’ | ’\t’ | ’\n’| ’\r’)   // Whitespace
     { $setType(Token.SKIP); } ; // Action: ignore
```

ANTLR Specifications for Scanners

Rules are names starting with a capital letter.

A character in single quotes matches that character.

```
LPAREN : ’(’ ;
```

A string in double quotes matches the string

```
IF : "if" ;
```

A vertical bar indicates a choice:

```
OP : ’+’ | ’-’ | ’*’ | ’/’ ;
```

ANTLR Specifications

Question mark makes a clause optional.

```
PERSON : ("wo")? ’m’ (’a’|’e’) ’n’ ;
```

(Matches man, men, woman, and women.)

Double dots indicate a range of characters:

```
DIGIT : ’0’..’9’;
```

Asterisk and plus match “zero or more,” “one or more.”

```
ID : LETTER (LETTER | DIGIT)* ;
NUMBER : (DIGIT)+ ;
```