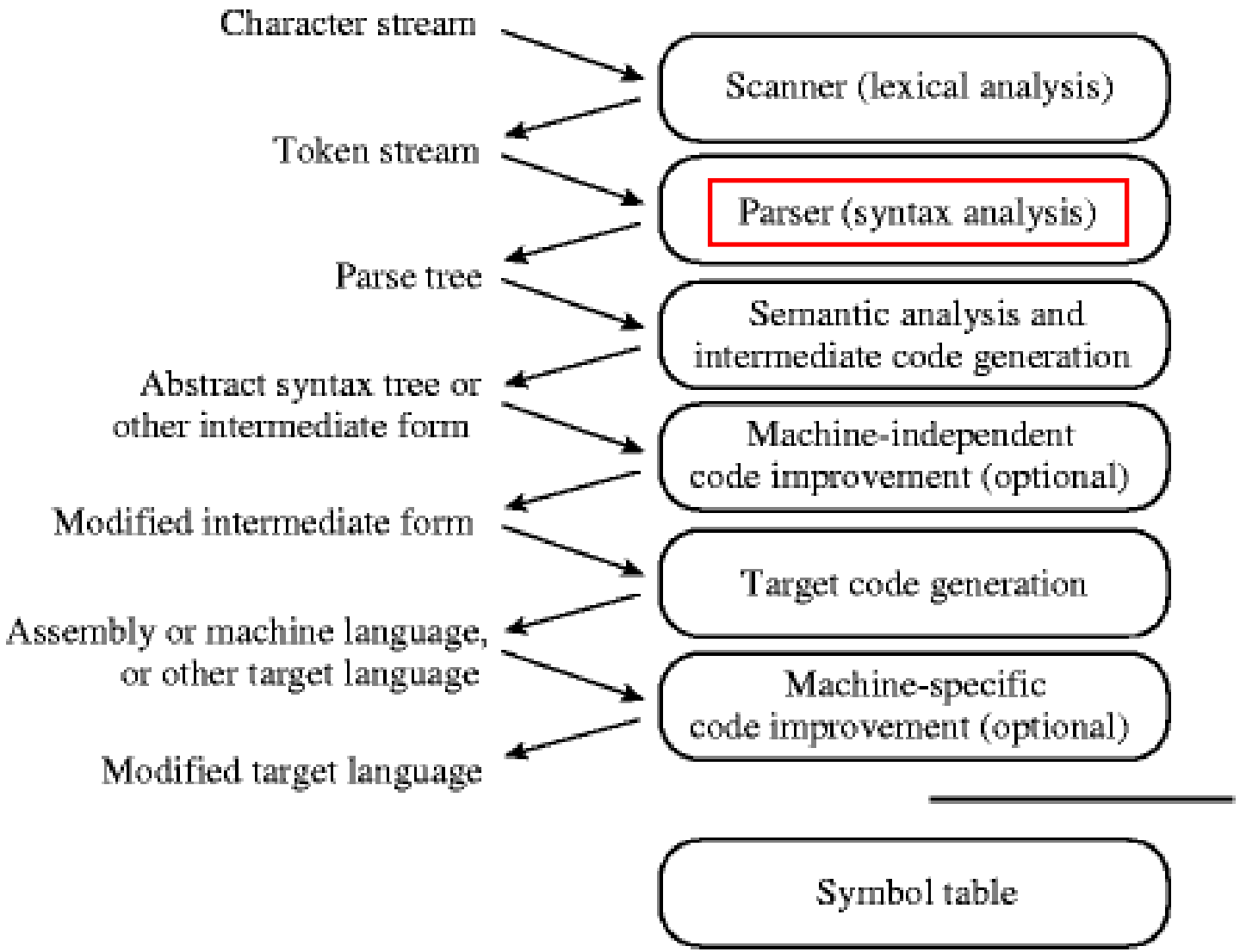
Syntax Specification
Regular Expressions

Phases of Compilation



Syntax Analysis

- Syntax:
  - Webster's definition: *1 a : the way in which linguistic elements (as words) are put together to form constituents (as phrases or clauses)*
- The syntax of a programming language
  - Describes its form
    - *i.e.* **Organization of tokens** *(elements)*
  - Formal notation
    - Context Free Grammars (CFGs)

Review: Formal definition of tokens

- A set of tokens is a set of strings over an alphabet
  - {read, write, +, -, *, /, :=, 1, 2, …, 10, …, 3.45e-3, …}
- A set of tokens is a *regular set* that can be defined by comprehension using a *regular expression*
- For every regular set, there is a *deterministic finite automaton* (DFA) that can recognize it
  - *i.e.* determine whether a string belongs to the set or not
  - Scanners extract tokens from source code in the same way DFAs determine membership

Regular Expressions

- A regular expression (RE) is:
  - A single character
  - The empty string, ε
  - The concatenation of two regular expressions
    - *Notation:* RE_1 RE_2 (*i.e.* RE_1 followed by RE_2)
  - The union of two regular expressions
    - *Notation:* $RE_1 \mid RE_2$
  - The closure of a regular expression
    - *Notation:* RE*
    - * is known as the *Kleene star*
    - * represents the concatenation of 0 or more strings
  - *Non-null enumeration*
    - *Notation:* RE^+
    - represents all non-null concatenations of RE (1 or more times)

Regular Expressions Basics

Let alphabet Σ={a,b} (means a and b are its only letters)

a^*=(λ, a, aa, aaa, ...}

(ab)*=(λ, ab, abab, ababab, ...}

a ∪ b=(a, λ, b, bb, bb, ...}

(a ∪ b)*= all strings containing a's and b's

(a*b*)*=(ab*)*= all strings containing a's and b's

$a^*b^*=\{a^ib^j \mid i >=0, j>=0)$

Building Regular Expressions

Regular Expressions as Language

- * while loop
  - iterates 0 or more times
- concatenation *uv*
  - sequential; first *u*, then *v*
- $u \cup v$ OR
  - select from one or the other or both

Description → Regular Expression

Let Σ={a,b} – all expressions over this alphabet

Strings with

- exactly one a b*ab*
- exactly two a's b*ab*ab*
- one or more a's (b*ab*)* or (a ∪ b)*a (a ∪ b)*
- even number of a's (b*ab*ab*)*
- even number of a's and exactly one b
- (aa)*b(aa)* ∪ (aa)*ab(aa)*a
- odd number of a's (b*ab*ab*)*b*ab*
- that don't contain aa (b ∪ ab)*(λ ∪ a)

Regular Expression → Description

Same alphabet

- (aa)* even number of a's
- (a ∪ b) (a ∪ b) (a ∪ b) (a ∪ b)

  all strings of length 4
- ((a ∪ b) (a ∪ b) (a ∪ b) (a ∪ b))*

  strings of length divisible by 4
- (aa)* ∩ ((a ∪ b) (a ∪ b) (a ∪ b) (a ∪ b))*

  strings of a's of length divisible by 4

Token Definition Example

- Numeric literals in Pascal
  - Definition of the token *unsigned_number*

digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

unsigned_integer → *digit digit** | *digit*$^{+}$

unsigned_number → *unsigned_integer* ((. *unsigned_integer*) | ε)
((e (+ | – | ε) *unsigned_integer*) | ε)

- **Recursion is not allowed in Regular Expressions!**

Exercise

digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

unsigned_integer → *digit digit**

unsigned_number → *unsigned_integer* ((. *unsigned_integer*) | ε)
((e (+ | – | ε) *unsigned_integer*) | ε)

- Regular expression for
  - Decimal numbers

number → …

Exercise

digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

unsigned_integer → *digit digit**

unsigned_number → *unsigned_integer* ((. *unsigned_integer*) | ε)
((e (+ | – | ε) *unsigned_integer*) | ε)

- Regular expression for
  - Decimal numbers

number → (+ | – | ε) *unsigned_integer* ((. *unsigned_integer*) | ε)

Exercise

digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

unsigned_integer → *digit digit**

unsigned_number → *unsigned_integer* ((. *unsigned_integer*) | ε)
((e (+ | – | ε) *unsigned_integer*) | ε)

- Regular expression for
  - Identifiers

identifier → …

Exercise

digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

unsigned_integer → *digit digit**

unsigned_number → *unsigned_integer* ((. *unsigned_integer*) | ε)
((e (+ | – | ε) *unsigned_integer*) | ε)

- Regular expression for
  - Identifiers

identifier → *letter* (*letter* | digit | _)*

letter → a | b | c | … | z