## Scanning

# Goal: Reading through source listing and identifying tuples of (TOKENTYPE, SrcValue) for the parser to use.

Scanner  $\equiv$  Lexer  $\equiv$  Lexical Analysis

Makes you wonder: "What is a lexicon?"

## Scanning

## Goal: Reading through source listing and identifying tuples of (TOKENTYPE, SrcValue) for the parser to use.

 $\text{Scanner} \equiv \text{Lexer} \equiv \text{Lexical Analysis}$ 

Makes you wonder: "What is a lexicon?"

The scanner **doesn't enforce syntax rules** (that's a parser's job).

So, what kind of errors does a scanner detect?

## Scanning

# Goal: Reading through source listing and identifying tuples of (TOKENTYPE, SrcValue) for the parser to use.

 $\text{Scanner} \equiv \text{Lexer} \equiv \text{Lexical Analysis}$ 

Makes you wonder: "What is a lexicon?"

The scanner **doesn't enforce syntax rules** (that's a parser's job).

So, what kind of errors does a scanner detect? (some examples:)

- Invalid character code (binary file? utf-8? not utf-8?)
- An invalid sequence of characters (my!name@domain.com) for a TOKENTYPE of EmailAddr)
- Missing characters (1.0e+ for a TOKENTYPE of FLOAT)

### The fundamental go-to tool for program source scanning are Regular Expressions

#### Given:

- A finite alphabet of characters  $\Sigma$
- ► The empty set Ø (required for completeness and closure proofs)
- λ an empty string

 $\lambda 
eq \emptyset \qquad \qquad \lambda 
ot \in \emptyset \qquad \qquad \emptyset 
ot \not\in \lambda$ 

 $\lambda$  is **also** a RE that matches **only** a zero length string (a little bit of a tautology).  $\epsilon$  is another common symbol, **our book uses**  $\lambda$  **so that's what we'll use**.

- The symbol  $s \in \Sigma$  (one character) is a RE that matches only *s*.
- A set of characters from Σ, T written in RE notation (a|b|c) that matches only one character of the input.

Written out in mathy set notation as  $T = \{a, b, c\}$  (as you would expect).

>  $\Sigma$  is itself a RE which matches **one** character from the alphabet. A period . is a convenient keyboard accessible synonym for the RE  $\Sigma$ .

Notice: sets are upper case (either Latin or Greek), characters are lower case!

Let A, B be REs and we define RE Operations (highest precedence last):

Alternation  $A|B = \{x \mid x \in A \text{ OR } x \in B\}$  x might be single characters or strings.  $x \in A$  means character sequence x is matched by RE A.

Concatentation  $AB = \{xy \mid x \in A, y \in B\}$ 

Kleene Closure  $A \star = A \star = \{\lambda\} \bigcup \{xA \star \mid x \in A\}$ 

Simply put: "zero or more As". Notice this is **postfix** notation (the operator comes after its argument(s)! We just use \* at the keyboard for \*

Grouping Parenthetical grouping overrides operator precedence (as expected).

Let *A*, *B* be REs and **we define RE Operations** (highest precedence last):

Alternation  $A|B = \{x \mid x \in A \text{ OR } x \in B\}$  x might be single characters or strings.  $x \in A$  means character sequence x is matched by RE A.

Concatentation  $AB = \{xy \mid x \in A, y \in B\}$ 

Kleene Closure  $A \star = A^* = \{\lambda\} \bigcup \{xA \star | x \in A\}$ Simply put: "zero or more As". Notice this is **postfix** notation (the operator comes after its argument(s)! We just use \* at the keyboard for \*

Grouping Parenthetical grouping overrides operator precedence (as expected).

Some examples with:  $\Sigma = \{a, b, c, \dots, z\}, X = \{f, g\}, Y = \{m\}, Z = \{s, t\}.$   $a\lambda$  matches "a"  $\lambda a$  matches "a"  $a|\lambda$  matches "a" or "" XYZ matches "fms", "fmt", "gms" or "gmt" X|Y|Z matches "f", "g", "m", "s" or "t" X(Y|Z) matches "fm", "gm", "fs", "ft", "gs" or "gt"  $X(Y|Z\star)$  matches "fm", "g", "fsts", "fssss", ...

Let *A*, *B* be REs and **we define RE Operations** (highest precedence last):

Alternation  $A|B = \{x \mid x \in A \text{ OR } x \in B\}$  x might be single characters or strings.  $x \in A$  means character sequence x is matched by RE A.

Concatentation  $AB = \{xy \mid x \in A, y \in B\}$ 

Kleene Closure  $A \star = A^* = \{\lambda\} \bigcup \{xA \star | x \in A\}$ Simply put: "zero or more As". Notice this is **postfix** notation (the operator comes after its argument(s)! We just use \* at the keyboard for  $\star$ 

Grouping Parenthetical grouping overrides operator precedence (as expected).

Some examples with:  $\Sigma = \{a, b, c, ..., z\}$ ,  $X = \{f, g\}$ ,  $Y = \{m\}$ ,  $Z = \{s, t\}$ . What strings do  $XY | Z \star$  match?

Let *A*, *B* be REs and **we define RE Operations** (highest precedence last):

Alternation  $A|B = \{x \mid x \in A \text{ OR } x \in B\}$  x might be single characters or strings.  $x \in A$  means character sequence x is matched by RE A.

Concatentation  $AB = \{xy \mid x \in A, y \in B\}$ 

Kleene Closure  $A \star = A^* = \{\lambda\} \bigcup \{xA \star | x \in A\}$ Simply put: "zero or more As". Notice this is **postfix** notation (the operator comes after its argument(s)! We just use \* at the keyboard for  $\star$ 

Grouping Parenthetical grouping overrides operator precedence (as expected).

Some examples with:  $\Sigma = \{a, b, c, ..., z\}$ ,  $X = \{f, g\}$ ,  $Y = \{m\}$ ,  $Z = \{s, t\}$ . What strings do  $XY | Z \star$  match?

Concatenation has higher precedence than alternation (|), so  $XY|Z \star \equiv (XY)|Z \star$  and matches:

"fm", "fmssssss", "gm", "fms", "fmt", "gms", "gmtttt", ...

## **Regular Expression Outcomes**

- Closure Given our definitions and operations, REs **are closed** any combination via these operators is another regular expression.
- Regular Set is the collection of character strings that are **generated** by a non-empty set of regular expressions.

"generated"  $\equiv$  "matches"  $\equiv$  "accepted"  $\equiv$  "detected"  $\equiv$  "verified"

Regular Language is a sequence of symbols whose **syntax** can be defined (generated) by a RE. Token Class (in the context of program source scanning, or "lexing") is a sequence of **characters** accepted by a RE.

## So many languages, let's begin our Taxonomy!...

Some languages can use REs not only for tokenization, but also for syntax verification (aka "parsing").

eg: many "config file" formats, many assembly languages.

Note that these steps don't use **the same RE**, in practice they use several different REs for tokenization and one RE for parsing.

Other languages (pretty much **all high level lanugages that you are most familier with**) use REs **just for tokenization**, and require more sophisticated grammars and language algorithms for parsing.

I know of no well known languages that require sophisticated grammars and language algorithms for tokenization.

(I'm sure there are some out there, but I'm not familiar with them...)

## ...a little sugar on top :)

Some common utility operations that are built on the fundamental RE operations (these would be lemmas in RE Theory)

Positive Closure  $A^+ = \{aA \star \mid a \in A\}$ 

Simply put: one or more instances of a regular set of A. When at the keyboard, simply type + and forego the superscript.

Inverse Character Sets Given  $\Pi$  a set of characters from  $\Sigma$ ,

```
Not(\Pi) = \{x \mid x \in \Sigma \text{ AND } x \notin \Pi\}
```

Written for many RE engines as [^abc].

Inverse Regular Sets Given A a RE,

 $Not(A) = \{x \mid x \in \{\text{strings from } \Sigma\} \text{ AND } x \notin A\}$ 

Finite Repetition  $A^k = \{x_1 x_2 \dots x_k | x_i \in A \text{ AND } k > 0\}$ 

There are several different ways various RE engines represent this at the keyboard.