Advanced Programming Languages. Practice 1.

EPIC Institute of Technology

September 22, 2023

1 Formal grammars

- Non-terminal symbols (A, B, …, Y, Z)
- Terminal symbols (a, b, …, y, z)
- Start non-terminal symbol (S)
- Set of rules:
  - $AcB \to \varepsilon$
  - $S \to CCCd$
  - $A \to a|bcD \quad \equiv \quad A \to a, A \to bcD$

1.1 Integers

$$S \to AB$$
$$A \to \varepsilon|+|-$$
$$B \to D|DB$$
$$D \to 0|1|2|3|4|5|6|7|8|9$$

Example: $S \to AB \to -B \to -DB \to -DD \to -4D \to -42.$

1.2 Brackets

$$S \to \varepsilon|(S)|SS$$

$$S \to \varepsilon|(S)S$$

Example: $S \to (S)S \to ()S \to ()(S)S \to ()((S)S)S \to ()(()S)S \to ()(()(S)S)S \to ()(()()S)S \to ()(()())S \to ()(()())$.

2 Types of grammars

Recursively enumerable (Type 0). no restrictions on grammars

Context-sensitive (Type 1). $\alpha A\beta \to \alpha\gamma\beta$.

Context-free (Type 2). $A \to \alpha$

Regular (Type 3). $A \to aB,\ A \to a$

2.1 Determine type of grammar

1.

$$\begin{gathered} S \to aBC \\ aB \to CD \\ C \to a \\ aC \to BD \\ D \to b \end{gathered}$$

2.

$$S \to \varepsilon|(S)S$$

3.

$$S \to \varepsilon|aSb$$

4.

$$\begin{gathered} S \to aSAB|aAB \\ BA \to BD \\ BD \to ED \\ ED \to AD \\ AD \to AB \\ aA \to ab \\ bA \to bb \\ bB \to bc \\ cB \to cc \end{gathered}$$

5.

$$\begin{gathered} S \to AB \\ A \to \varepsilon|+|- \\ B \to D|DB \\ D \to 0|1|2|3|4|5|6|7|8|9 \end{gathered}$$

Can we turn it into a regular one?

$$\begin{gathered} S \to \varepsilon B|+B|-B \\ B \to 0|1|2|3|4|5|6|7|8|9|0B|1B|2B|3B|4B|5B|6B|7B|8B|9B \end{gathered}$$

3 Regular expressions

- $[abcd]$ — one of the characters from the set $\{a, b, c, d\}$
- a — single character
- (R) — group
- $*$, $+$, $?$ — repetition
- $R|R$ — boolean or

Examples:

- `first|second`
- `[+-]?[0-9]+`
- `(Mr.  |Mrs.  )?[A-Z][a-z]+ [A-Z][a-z]+`
- `[+-]?([1-9][0-9]+|0x[0-9a-fA-F]+|0[0-7]*)`

4 Backus–Naur form

```
<expr> ::= <sum>
<sum> ::= <mul> | <sum> "+" <mul> | <sum> "-" <mul>
<mul> ::= <last> | <mul> "*" <last> | <mul> "/" <last>
<last> ::= <number> | "+" <last> | "-" <last> | "(" <expr> ")"
<number> ::= <digit> | <digit> <number>
<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

What kind of grammar can we specify?
Let's try to parse:
$2 * 4 + (2 + -3 * 5)/7$

Syntax for regular expressions above:

```
<regex> ::= <concat> | <regex> "|" <concat>
<concat> ::= <item> | <concat> <item>
<item> ::= <item> <repetition> | "(" <item> ")" | <character> | <group>
<group> ::= "[" <groupItems> "]"
<groupItems> ::= "" | <character> <groupItems>
<character> ::= "a" | "b" | ... | "z" | "A" | ... "Z"
<repetition> ::= "?" | "+" | "*"
```