From RE to Scanner (1)

Token Type (CharCat)

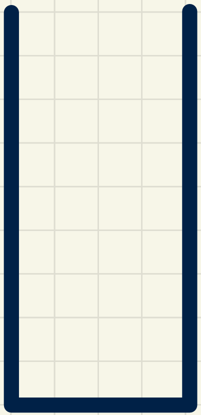
r	0, 1, 2, ..., 9	EOF	**Other**
Register	*Digit*	*Other*	*Other*

Transition

	Register	***Digit***	***Other***
s_0	s_1	s_e	s_e
s_1	s_e	s_2	s_e
s_2	s_e	s_2	s_e
s_e	s_e	s_e	s_e

Token Type (Type)

s_0	s_1	s_2	s_e
invalid	*invalid*	*register*	*invalid*



Regular Expression: r[0..9]+

```
NextWord()
  -- Stage 1:  Initialization
  state := s0 ; word := ε
  initialize an empty stack s ; s.push(bad)
  -- Stage 2:  Scanning Loop
  while (state ≠ se)
    NextChar(char) ; word := word + char
    if state ∈ F then reset stack s end
    s.push(state)
    cat := CharCat[char]
    state := δ[state, cat]
  -- Stage 3:  Rollback Loop
  while (state ∉ F ∧ state ≠ bad)
    state := s.pop()
    truncate word
  -- Stage 4:  Interpret and Report
  if state ∈ F then return Type[state]
  else return invalid
  end
```

Example input: r24

word:

state:

cat:

From RE to Scanner (2)

Regular Expression: r[0..9]+

Token Type (CharCat)

r	0, 1, 2, …, 9	EOF	**Other**
Register	*Digit*	*Other*	*Other*

Transition

	Register	***Digit***	***Other***
s_0	s_1	s_e	s_e
s_1	s_e	s_2	s_e
s_2	s_e	s_2	s_e
s_e	s_e	s_e	s_e

Token Type (Type)

s_0	s_1	s_2	s_e
invalid	*invalid*	*register*	*invalid*

```
NextWord()
  -- Stage 1:  Initialization
  state := s0 ; word := ϵ
  initialize an empty stack s ; s.push(bad)
  -- Stage 2:  Scanning Loop
  while (state ≠ se)
    NextChar(char) ; word := word + char
    if state ∈ F then reset stack s end
    s.push(state)
    cat := CharCat[char]
    state := δ[state, cat]
  -- Stage 3:  Rollback Loop
  while (state ∉ F ∧ state ≠ bad)
    state := s.pop()
    truncate word
  -- Stage 4:  Interpret and Report
  if state ∈ F then return Type[state]
  else return invalid
  end
```

Example input: r24*3

word:

state:

cat:

Context-Free Grammar (CFG): Terminology

The following language that is ***non-regular***

$$\{0^n \# 1^n \mid n \geq 0\}$$

can be described using a ***context-free grammar (CFG)***:

$$\begin{array}{rcl} A & \rightarrow & 0A1 \\ A & \rightarrow & B \\ B & \rightarrow & \# \end{array}$$

Visualization Derivations from CFG

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

- Shortest Derivation?
- 000#111?
- 010#101?

Discussion: Compare Two CFGs

Expression	→	*IntegerConstant*
	|	*BooleanConstant*
	|	*BinaryOp*
	|	*UnaryOp*
	|	(*Expression*)
IntegerConstant	→	*Digit*
	|	*Digit IntegerConstant*
	|	− *IntegerConstant*
Digit	→	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
BooleanConstant	→	`TRUE`
	|	`FALSE`

v1

BinaryOp	→	*Expression* + *Expression*
	|	*Expression* − *Expression*
	|	*Expression* * *Expression*
	|	*Expression* / *Expression*
	|	*Expression* && *Expression*
	|	*Expression* || *Expression*
	|	*Expression* => *Expression*
	|	*Expression* == *Expression*
	|	*Expression* /= *Expression*
	|	*Expression* > *Expression*
	|	*Expression* < *Expression*
UnaryOp	→	! *Expression*

v2

ArithmeticOp	→	*ArithmeticOp* + *ArithmeticOp*
	|	*ArithmeticOp* − *ArithmeticOp*
	|	*ArithmeticOp* * *ArithmeticOp*
	|	*ArithmeticOp* / *ArithmeticOp*
	|	(*ArithmeticOp*)
	|	*IntegerConstant*
RelationalOp	→	*ArithmeticOp* == *ArithmeticOp*
	|	*ArithmeticOp* /= *ArithmeticOp*
	|	*ArithmeticOp* > *ArithmeticOp*
	|	*ArithmeticOp* < *ArithmeticOp*
LogicalOp	→	*LogicalOp* && *LogicalOp*
	|	*LogicalOp* || *LogicalOp*
	|	*LogicalOp* => *LogicalOp*
	|	! *LogicalOp*
	|	(*LogicalOp*)
	|	*RelationalOp*
	|	*BooleanConstant*

Context-Free Grammar (CFG): Example Version 1

Example: (1 + 2) => (5 / 4)

Expression	$\rightarrow$	*IntegerConstant*
	|	*BooleanConstant*
	|	*BinaryOp*
	|	*UnaryOp*
	|	(*Expression*)
IntegerConstant	$\rightarrow$	*Digit*
	|	*Digit IntegerConstant*
	|	−*IntegerConstant*
Digit	$\rightarrow$	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
BooleanConstant	$\rightarrow$	`TRUE`
	|	`FALSE`

BinaryOp	$\rightarrow$	*Expression* + *Expression*
	|	*Expression* − *Expression*
	|	*Expression* $\star$ *Expression*
	|	*Expression* / *Expression*
	|	*Expression* && *Expression*
	|	*Expression* || *Expression*
	|	*Expression* => *Expression*
	|	*Expression* == *Expression*
	|	*Expression* /= *Expression*
	|	*Expression* > *Expression*
	|	*Expression* < *Expression*
UnaryOp	$\rightarrow$	! *Expression*

Context-Free Grammar (CFG): Example Version 1

Example: 3 * 5 + 4

Expression	→	*IntegerConstant*
	|	*BooleanConstant*
	|	*BinaryOp*
	|	*UnaryOp*
	|	(*Expression*)
IntegerConstant	→	*Digit*
	|	*Digit IntegerConstant*
	|	−*IntegerConstant*
Digit	→	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
BooleanConstant	→	`TRUE`
	|	`FALSE`

BinaryOp	→	*Expression* + *Expression*
	|	*Expression* − *Expression*
	|	*Expression* ∗ *Expression*
	|	*Expression* / *Expression*
	|	*Expression* && *Expression*
	|	*Expression* || *Expression*
	|	*Expression* => *Expression*
	|	*Expression* == *Expression*
	|	*Expression* /= *Expression*
	|	*Expression* > *Expression*
	|	*Expression* < *Expression*
UnaryOp	→	! *Expression*

Context-Free Grammar (CFG): Example Version 2

Expression	$\rightarrow$	*ArithmeticOp*
	|	*RelationalOp*
	|	*LogicalOp*
	|	(*Expression*)
IntegerConstant	$\rightarrow$	*Digit*
	|	*Digit IntegerConstant*
	|	−*IntegerConstant*
Digit	$\rightarrow$	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
BooleanConstant	$\rightarrow$	TRUE
	|	FALSE

Example: (1 + 2) => (5 / 4)

ArithmeticOp	$\rightarrow$	*ArithmeticOp* + *ArithmeticOp*
	|	*ArithmeticOp* − *ArithmeticOp*
	|	*ArithmeticOp* * *ArithmeticOp*
	|	*ArithmeticOp* / *ArithmeticOp*
	|	(*ArithmeticOp*)
	|	*IntegerConstant*
RelationalOp	$\rightarrow$	*ArithmeticOp* == *ArithmeticOp*
	|	*ArithmeticOp* /= *ArithmeticOp*
	|	*ArithmeticOp* > *ArithmeticOp*
	|	*ArithmeticOp* < *ArithmeticOp*
LogicalOp	$\rightarrow$	*LogicalOp* && *LogicalOp*
	|	*LogicalOp* || *LogicalOp*
	|	*LogicalOp* => *LogicalOp*
	|	! *LogicalOp*
	|	(*LogicalOp*)
	|	*RelationalOp*
	|	*BooleanConstant*

Context-Free Grammar (CFG): Example Version 2

Expression	→	*ArithmeticOp*
	|	*RelationalOp*
	|	*LogicalOp*
	|	(*Expression*)
IntegerConstant	→	*Digit*
	|	*Digit IntegerConstant*
	|	−*IntegerConstant*
Digit	→	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
BooleanConstant	→	`TRUE`
	|	`FALSE`

ArithmeticOp	→	*ArithmeticOp* + *ArithmeticOp*
	|	*ArithmeticOp* − *ArithmeticOp*
	|	*ArithmeticOp* ⋆ *ArithmeticOp*
	|	*ArithmeticOp* / *ArithmeticOp*
	|	(*ArithmeticOp*)
	|	*IntegerConstant*
RelationalOp	→	*ArithmeticOp* == *ArithmeticOp*
	|	*ArithmeticOp* /= *ArithmeticOp*
	|	*ArithmeticOp* > *ArithmeticOp*
	|	*ArithmeticOp* < *ArithmeticOp*
LogicalOp	→	*LogicalOp* && *LogicalOp*
	|	*LogicalOp* || *LogicalOp*
	|	*LogicalOp* => *LogicalOp*
	|	! *LogicalOp*
	|	(*LogicalOp*)
	|	*RelationalOp*
	|	*BooleanConstant*

Q: No **semantic analysis** at all for Version 2 grammar?

[redacted]

Context-Free Grammar (CFG): Example Version 2

Example: 3 * 5 + 4

```
Expression       →  ArithmeticOp
                 |  RelationalOp
                 |  LogicalOp
                 |  ( Expression )

IntegerConstant  →  Digit
                 |  Digit IntegerConstant
                 |  -IntegerConstant

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

BooleanConstant  →  TRUE
                 |  FALSE
```

```
ArithmeticOp  →  ArithmeticOp + ArithmeticOp
              |  ArithmeticOp − ArithmeticOp
              |  ArithmeticOp * ArithmeticOp
              |  ArithmeticOp / ArithmeticOp
              |  (ArithmeticOp)
              |  IntegerConstant
RelationalOp  →  ArithmeticOp == ArithmeticOp
              |  ArithmeticOp /= ArithmeticOp
              |  ArithmeticOp > ArithmeticOp
              |  ArithmeticOp < ArithmeticOp
LogicalOp     →  LogicalOp && LogicalOp
              |  LogicalOp || LogicalOp
              |  LogicalOp => LogicalOp
              |  ! LogicalOp
              |  (LogicalOp)
              |  RelationalOp
              |  BooleanConstant
```

CFG: Formal Definition

Design the CFG for strings of properly-nested parentheses.

e.g., ( ) , ( ) ( ) , ((( ) ( ))) ( ) , *etc.*

Present your answer in a formal manner.

A ***context-free grammar (CFG)*** is a 4-tuple (V, Σ, R, S):

- V is a finite set of **variables**.
- Σ is a finite set of ***terminals***. [redacted]
- R is a finite set of ***rules*** s.t.

$$R \subseteq \{v \rightarrow s \mid \text{[redacted]}\}$$

- $S \in V$ is is the **start variable**.

Given strings u, v, w , variable a rule :

- $\boxed{uAv \Rightarrow uwv}$ menas that uAv ***yields*** uwv.
- $\boxed{u \overset{*}{\Rightarrow} v}$ means that u ***derives*** v, if:
  - $u = v$; or
  - $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v$ [a ***yield sequence***]

Given a CFG $G = (V, \Sigma, R, S)$, the language of G

$$L(G) = \{\text{[redacted]}\}$$

Context-Free Grammar (CFG): Example Version 3

$$
\begin{array}{lcl}
Expr & \rightarrow & Expr + Term \\
 & | & Term \\
Term & \rightarrow & Term * Factor \\
 & | & Factor \\
Factor & \rightarrow & (Expr) \\
 & | & \mathtt{a}
\end{array}
$$

Example: a * a + a

Context-Free Grammar (CFG): from RE (1)

RE	CFG
$L(\ \epsilon\ )$	
$L(\ a\ )$	
$L(\ E + F\ )$	
$L(\ EF\ )$	
$L(\ E^*\ )$	
$L(\ (E)\ )$	

Context-Free Grammar (CFG): from RE (2)

$(0 + 1)^*1(0+1)$

$(00 + 1)^* + (11 + 0)^*$

Context-Free Grammar (CFG): from DFA