What is the corresponding regex?

((1-9)|1[0-2]):[0-5][0-9](AM|PM)?

What does the above match?

- Matches clock time, may or may not be told if it is AM or PM.
More CFG Notation

- compact representation of alternative rules using vertical bar
- e.g., rewrite Rules 8 and 9:
  \[ \text{<noun>} \rightarrow \text{man} \mid \text{dog} \]
- empty rule
  \[ \text{<article>} \rightarrow \varepsilon \]

- Remember: to derive any input string from the rules, the systematic approach is to always expand the left-most non-terminal
Arithmetic Expressions

• build toy calculator for integer arithmetic
  • addition, subtraction, multiplication, division, parentheses
• need grammar (tokens, variables, rules)
• tokens:
  • integer constants
  • +, -, *, /, (, )
• start with intuitive grammar
Arithmetic CFG

Tokens: +, -, *, /, (, ), and integer constants

Start nonterminal: <expression>

- `<expression>` → `<term>` `<moreTerms>`
- `<term>` → `<factor>` `<moreFactors>`
- `<factor>` → ( `<expression>` ) | int
- `<moreTerms>` → ε | + `<term>` `<moreTerms>` | − `<term>` `<moreTerms>`
- `<moreFactors>` → ε | * `<factor>` `<moreFactors>` | / `<factor>` `<moreFactors>`

What is the derivation for 7+34
A Simple Derivation

- leftmost derivation for 7 + 34

<expression>
=> <term> <moreTerms>          (R1)
=> <factor> <moreFactors> <moreTerms> (R2)
=> int <moreFactors> <moreTerms> (R4)
=> int <moreTerms> (R8)
=> int + <term> <moreTerms> (R6)
=> int + <factor> <moreFactors> <moreTerms> (R2)
=> int + int <moreFactors> <moreTerms> (R4)
=> int + int <moreTerms> (R8)
=> int + int (R5)
Arithmetic CFG – Shortened

\[
\begin{align*}
&E \rightarrow &T & MT \\
&T \rightarrow &F & MF \\
&F \rightarrow & ( & E ) \\
& & & \text{int} \\
&MT \rightarrow & \epsilon \\
& & & + & T & MT \\
& & & - & T & MT \\
&MF \rightarrow & \epsilon \\
& & & * & F & MF \\
& & & / & F & MF
\end{align*}
\]
Parse Trees

- also called derivation trees
- visualize entire derivation at once
- non-terminal are internal nodes
  - start symbols is root of tree
  - children of nodes are given by derivation rule
- terminals are leaf nodes
  - show value of terminal
Derivation Tree

- derivation tree for “7 + 34”
Unambiguous Grammar

- We want our grammar to be unambiguous: an implicit order of evaluating expressions in the absence of parentheses
- For that there are two rules: Precedence Rules and Associativity Rules
Precedence

- Precedence: grouping symbols with different priorities
  - example: $6 + 3 \times 4$
  - read as $(6 + 3) \times 4$ or $6 + (3 \times 4)$?

Let’s say these are the rules of my grammar:

```
<Expr>  ->  <Expr> + <Expr>
<Expr>  ->  <Expr> * <Expr>
<Expr>  ->  int
```

This has two possible parse trees that may be created by the parser.
Precedence

To remove ambiguity, the rules of the grammar need to be modified:

- `<Expr>  ->  <Expr1>`
- `<Expr>  ->  <Expr>  +  <Expr1>`
- `<Expr1>  ->  <Expr1>  *  <Expr1>`
- `<Expr1>  ->  int`
Associativity

- associativity: grouping symbols with same precedence/priority
  - example: 6 / 3 / 2
  - Can be read as (6/3)/4 or 6/(3/4)?
- The grammar should be left associative or right associative
- For example, here is a grammar with ambiguous associativity

```
<Expr> -> <Expr> / <Expr>
<Expr> -> int
```
Associativity

To remove associativity ambiguity, and make this left associative, the rules of the grammar need to be modified:

<Expr>  ->  <Expr> / <Expr1>
<Expr>  ->  <Expr1>
<Expr1> ->  int

How will I make it right associative?
• Ultimately it is impossible to know if a grammar is unambiguous or not! This is why creating the rules to get rid of as many ambiguities as possible is very important!

• If our specification is unambiguous, we know which tree to produce without ambiguity

• After constructing the tree, the parser evaluates it by traversing the tree.
Parse Tree - Evaluation

- Evaluation gives the value of the expression by traversing the tree

- Post-order traversal
  - recursive tree traversal algorithm
  - evaluate node after evaluating children
  - If you are looking at node N, evaluate the left subtree, then the right subtree, then the node.
Post-order Traversal

Postorder (Left, Right, Root) : 4 5 2 3 1