Tutorial 1 and Tutorial 2

Sudakshina Dutta

IIT Goa

27^{th} February, 2022

Tutorial 1

Write regular expression for all strings with no repeated digit

Tutorial 1

Write regular expression for all strings with no repeated digit

- $0?1?2?3?4?5?6?7?8?9? \mid 1?0?2?3?4?5?6?7?8?9? \mid \cdots$

Tutorial 1

All strings of digits with at most one repeated digits

Tutorial 2-4.3.1

The following is a grammar for regular expressions over symbols *a* and *b* only, using $+$ in place of j for union, to avoid conflict with the use of vertical bar as a metasymbol in grammars:

- $rexpr \rightarrow rexpr + rterm | rterm$
- $rterm \rightarrow rterm\ rfactor | rfactor$
- $rfactor \rightarrow rfactor * | rprimary$
- $rprimary \rightarrow a | b$

1. Left factor this grammar.
2. Does left factoring make the grammar suitable for top-down parsing ?
3. In addition to left factoring, eliminate left recursion from the original grammar.
4. Is the resulting grammar suitable for top-down parsing?

Tutorial 2-4.3.1

The following is a grammar for regular expressions over symbols a and b only, using $+$ in place of j for union, to avoid conflict with the use of vertical bar as a metasymbol in grammars:

- $rexpr \rightarrow rexpr + rterm | rterm$
- $rterm \rightarrow rterm\ rfactor | rfactor$
- $rfactor \rightarrow rfactor * | rprimary$
- $rprimary \rightarrow a | b$

1. Left factor this grammar.
    - The same grammar is generated

We don't have the Production of the Form:
A -> @ (B1) | @ (B2)

Tutorial 2-4.3.1

The following is a grammar for regular expressions over symbols a and b only, using $+$ in place of j for union, to avoid conflict with the use of vertical bar as a metasymbol in grammars:

- $rexpr \rightarrow rexpr + rterm | rterm$
- $rterm \rightarrow rterm\ rfactor | rfactor$
- $rfactor \rightarrow rfactor * | rprimary$
- $rprimary \rightarrow a | b$

1. Does left factoring make the grammar suitable for top-down parsing ? No

Tutorial 2-4.3.1

The following is a grammar for regular expressions over symbols a and b only, using $+$ in place of j for union, to avoid conflict with the use of vertical bar as a metasymbol in grammars:

- $rexpr \rightarrow rexpr + rterm | rterm$
- $rterm \rightarrow rterm\ rfactor | rfactor$
- $rfactor \rightarrow rfactor * | rprimary$
- $rprimary \rightarrow a | b$

Don't get confused. Here
+ is normal Addition.
| is OR as usual

1. In addition to left factoring, eliminate left recursion from the original grammar.
   - $rexpr \rightarrow rterm\ rexpr'$
   - $rexpr' \rightarrow +rterm\ rexpr' | \epsilon$
   - $rterm \rightarrow rfactor\ rterm'$
   - $rterm' \rightarrow rfactor\ rterm' | \epsilon$
   - $rfactor \rightarrow rprimary\ rfactor'$
   - $rfactor' \rightarrow *\ rfactor' | \epsilon$
   - $rprimary \rightarrow a | b$

We need to check whether this grammar is LL(1) or not

Tutorial 2-4.3.1

Computing FIRST & FOLLOW.

- $rexpr \rightarrow rterm\ rexpr'$
- $rexpr' \rightarrow +rterm\ rexpr'|\epsilon$
- $rterm \rightarrow rfactor\ rterm'$
- $rterm' \rightarrow rfactor\ rterm'|\epsilon$
- $rfactor \rightarrow rprimary\ rfactor'$
- $rfactor' \rightarrow *\ rfactor'|\epsilon$
- $rprimary \rightarrow a|b$

$FIRST(rexpr) = FIRST(rterm) = FIRST(rfactor) = FIRST(rprimary)$
$= \{a, b\}$
$FIRST(rexpr') = \{+, \epsilon\}$
$FIRST(rterm') = \{a, b, \epsilon\}$
$FIRST(rfactor') = \{*, \epsilon\}$
$FOLLOW(rexpr) = FOLLOW(rexpr') = \{\$\}$
$FOLLOW(rterm) = \{+, \$\} = FOLLOW(rterm')$
$FOLLOW(rfactor) = \{a, b, +, \$\}$
$FOLLOW(rfactor') = \{a, b, +, \$\}$
$FOLLOW(rprimary) = \{a, b, +, \$, *\}$

Tutorial 2-4.3.1

Constructing Predictive Parsing Table.

	a	b	$+$	$*$	$\$$
$rexpr$	$rexpr \to rterm\ rexpr'$	$rexpr \to rterm\ rexpr'$			
rexpr'			$rexpr' \to +rterm$		$rexpr' \to \epsilon$
$rterm$	$rterm \to rfactor\ rterm'$	$rterm \to rfactor\ rterm'$			
$rterm'$	$rterm' \to rfactor\ rterm'$	$rterm' \to rfactor\ rterm'$	$rterm' \to \epsilon$		$rterm' \to \epsilon$
$rfactor$	$rfactor \to rprimary\ rfactor'$	$rfactor \to rprimary\ rfactor'$			
$rfactor'$				$rfactor' \to *\ rfactor'$	
$rprimary$	$rprimary \to a$	$rprimary \to b$			

Since there is no repeated entry, It is LL(1) grammar. So it is suitable for TopDownParsing/PredictiveParsing/LL(1)Parsing.

Tutorial 2-4.4.6

- A grammar is ϵ-free if no production body is ϵ (called an ϵ-production).
  - Give an algorithm to convert any grammar into an ϵ-free grammar that generates the same language (with the possible exception of the empty string. No ϵ-free grammar can generate ϵ). Hint : First find all the non-terminals that are nullable, meaning that they generate ϵ, perhaps by a long derivation.
  - Apply your algorithm to the grammar $S \rightarrow aSbS|bSaS|\epsilon$

Tutorial 2-4.4.6

Input : A grammar $G = \langle N, T, P, S \rangle$ which has some epsilon productions
Output : A grammar $G' = \langle N, T, P', S \rangle$ which has some epsilon productions

1. Find out the non-terminals which are nullable. Let the set of such nullable non-terminals be N_n
2. Let $A \in N - N_n$. There is a production $A \rightarrow \alpha\beta \in P$ and $\alpha, \beta \in N_n$. Add $A \in N_n$. Continue step 2 until no more non-terminal from the set $N - N_n$ can be added to N_n
3. Initialize the set of productions P' with the productions of the for $B \rightarrow \gamma\delta$ such that $\gamma, \delta \notin N_n$
4. Consider a member $x \in N_n$. Let there be a production $B \rightarrow \gamma x \delta \in P$ where γ, δ can be any symbol. Add both $B \rightarrow \gamma x \delta$ and $B \rightarrow \gamma\delta$ to P'. Continue step 4 until no more production can be added to P'

Tutorial 2-4.4.6

- A grammar is ϵ-free if no production body is ϵ (called an ϵ-production).
  - Apply your algorithm to the grammar $S \rightarrow aSbS|bSaS|\epsilon$

The set of nullable non-terminals N_n is $\{S\}$. The $\epsilon-$free grammar G' is $\langle N, T, P', S\rangle$ where P' is
$S \rightarrow aSbS|bSaS|abS|aSb|baS|bSa|ab|ba$