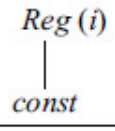
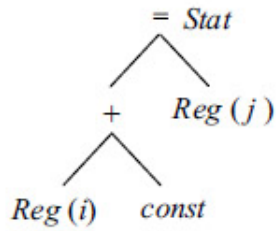
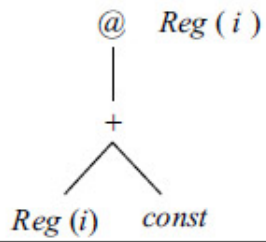
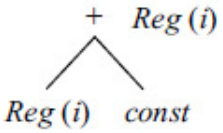
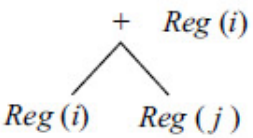
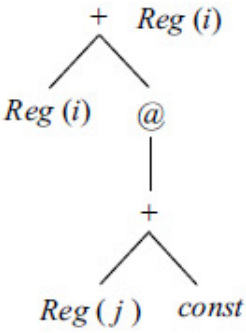
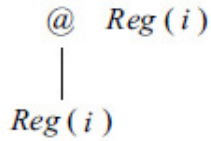
1. Для регулярного выражения **c(bc)*|(ba*c|d*)b** над алфавитом T = **{a, b, c, d}** построить эквивалентный минимальный детерминированный конечный автомат. Для полученного автомата построить эквивалентную праволинейную грамматику.
2. Для регулярного выражения **(a(c*|b)c)*|ad*** над алфавитом T = **{a, b, c, d}** построить эквивалентный минимальный детерминированный конечный автомат. Для полученного автомата построить эквивалентную праволинейную грамматику.
3. Для грамматики **G = {{A, S, B, C}, {a, b, c}, {S → ABC | AC, B → Bb | b, A → a, C → a | ε}, S}** написать эквивалентную LL(1) грамматику G_1. Для грамматики G_1 построить LL(1) анализатор и продемонстрировать его работу на цепочке **aa**.
4. Является ли грамматика **G = {{A, S, B}, {a, c}, {S → BaA | B, B → Bc | ε, A → c}, S}** LR(1)-грамматикой? Построить детерминированный правый анализатор. Является ли G LR(0) или LL(1) грамматикой? Продемонстрировать работу анализатора на цепочке **cac**.
5. Для логического выражения **(*A* and *B* or *C*) or not (*D* or *E*)** сгенерировать код на командах перехода и изобразить атрибутированное дерево.
6. Для арифметического выражения ***A* * (*B* + *C* * *D*)+ *E* * *F*** с помощью алгоритма Сети-Ульмана сгенерировать код и изобразить атрибутированное дерево.
7. Для оператора присваивания **a= d[7 + b[i]] + 2** сгенерировать оптимальный код методом сопоставления образцов (образцы на следующей странице)

№	Образец	Правило грамматики	Команда / стоимость	
1		*Reg* → *const*	`MOVE #const,Ri`	2
2		*Stat* → '=' '+' *Reg const Reg*	`MOVE Rj, const(Ri)`	4
3		*Reg* → '@' +' *Reg const*	`MOVE const(Ri),Ri`	4
4		*Reg* → '+' *Reg const*	`ADD #const,Ri`	3
5		*Reg* → '+' *Reg Reg*	`ADD Rj,Ri`	2
6		*Reg* → '+' *Reg* '@' '+' *Reg const*	`ADD const(Rj),Ri`	4
7		*Reg* → '@' *Reg*	`MOVE (Ri),Ri`	2