Вариант 1

Для грамматики G[E] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[E]:

1. E → TA
2. A → ε | + TA | - TA
3. T → OB
4. B → ε | *OB | /OB
5. O → num | id | (E)

num – числовая константа Ц{Ц}, id – идентификатор Б{Б|Ц}, Б – {a, b, c, ...z, A, B, …, Z}, Ц – {0, 1, …, 9}.

Вариант 2

Для грамматики G[<условный оператор>] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[<условный оператор>]:

1. <условный оператор> ::= IF <условие> THEN <оператор>
2. <условие> ::= <выражение> <операция отношения> <выражение>
3. <выражение> ::= <терм> {+ <терм>}
4. <терм> ::= <множитель> {* <множитель>}
5. <множитель> ::= < идентификатор > | (<выражение>)
6. <идентификатор> ::= <буква> {<буква> | <число>}
7. <оператор> ::= <выражение> | <условный оператор>

<операция отношения> ::= ”==” | ”<” | ”<=” | ”>” | ”>=” | ”!=”

<буква> ::= a | b | c | … | z | A | B | … | Z

<число> ::= <цифра> {<цифра>}

<цифра> 0 | 1 |...| 8 | 9

Вариант 3

Для грамматики G[E] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[E]:

1. E -> E + T | E - T | T
2. T -> T * F | T / F | F
3. F -> V ^ F | V
4. V -> (E) | id | number | ε

number – числовая константа Ц{Ц}

id – идентификатор Б{Б|Ц}

Б – {a, b, c, ...z, A, B, …, Z}, Ц – {0, 1, …, 9}

Вариант 4

Для грамматики G[S] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[S]:

1. S → select X from X
2. X → AY
3. Y → ,AY | ε
4. A → letter {letter}

letter → {a, b, c, ...z, A, B, …, Z}

Примечание: Данная грамматика для упрощенного варианта SQLзапроса «select»: «select столбец1, столбец2, … from таблица1, таблица2, …»

Вариант 5

Для грамматики G[<E>] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[<E>]:

1. E → E1 | E1 [= | < | >] E1
2. E1 → T { [+ | - | “or”] T }
3. T → F { [* | / | “and”] F }
4. F → “true” | “false” | “not” F | (E)

Вариант 6

Для грамматики G[<Выражение>] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[<Выражение>]:

1. <Выражение> = <Слагаемое> { (+|-) <Слагаемое> }
2. <Слагаемое> = <Множитель> { (*|/) <Множитель> }
3. <Множитель> = [+|-] <ДробноеЧисло> | <Функция> |(<Выражение>)
4. <ДробноеЧисло> = <ЦелаяЧасть> | <ЦелаяЧасть> .<ДробнаяЧасть>
5. <ЦелаяЧасть> = <Цифра>{<Цифра>}
6. <ДробнаяЧасть> = <Цифра>{<Цифра>}
7. <Функция> = <ИмяФункции>(<Выражение>)

<ИмяФункции> = “sin” | “cos” | “tg” | “ctg” | “log” | “ln”

<Цифра> = 0|1|2|3|4|5|6|7|8|9

Вариант 7

Для грамматики G[<expr>] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[<expr>]:

1. <expr> ::= <expr_add_sub> | <expr><operator_mul_div> <expr_add_sub>
2. <expr_add_sub> ::= <term> | <expr_add_sub><operator_mul_div> <term>
3. <term> ::= <integer> | "(" <expr> ")"
4. <operator_mul_div> ::= "*" | "/" | "%"
5. <operator_add_sub> ::= "+" | "-"
6. <integer> ::= <digit>|<integer><digit>

<digit> ::= 0|1|2|3|4|5|6|7|8|9

Вариант 8

Для грамматики G[<число>] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[<число>]:

1. <число> ::= [<знак>] <целая часть> . [<дробная часть>] [E [<знак>] <порядок>] | [<знак>] [<целая часть>] . <дробная часть> [E [<знак>] <порядок>]
2. <целая часть> ::= <цифра> [<целая часть>]
3. <дробная часть> ::= <целая часть>
4. <порядок> ::= <целая часть>

<цифра> ::= 0|1|2|3|4|5|6|7|8|9

<знак> ::= +|-

Вариант 9

Для грамматики G[<Выражение>] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[<Выражение>]:

1. <Выражение> = <Простое выражение> [<Операция отношения> <Простое
2. выражение>]
3. <Простое выражение> = <Терм> {<Аддитивная операция> <Терм>}
4. <Терм> = <Фактор> {<Мультипликативная операция> <Фактор>}
5. <Фактор> = <Переменная> | “not” <Фактор> | “(” <Выражение> “)”
6. <Переменная> = Буква {Буква | Цифра}
7. <Операция отношения> = “=” | “!=” | “<” | “<=” | “>” | “>=”
8. <Аддитивная операция> = “+” | “-” | “or”
9. <Мультипликативная операция> = “*" | “/" | “%"

Буква – [a, b, c, ...z, A, B, …, Z], Цифра – [0, 1, …, 9]

Вариант 10

Для грамматики G[<ФОРМУЛА>] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[<ФОРМУЛА>]:

1. <ФОРМУЛА> →<ФОРМУЛА> <ЗНАК> <ФОРМУЛА> | <ЧИСЛО> | '(' <ФОРМУЛА> ')'
2. <ЧИСЛО' → <ЦИФРА> | <ЧИСЛО><ЦИФРА>

<ЗНАК> → '+' | '-' | '*' | '/'

<ЦИФРА> → '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

Вариант 11

Для грамматики G[S] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[S]:

1. S → NP VP
2. NP → Pro | PropN | Det A Nom | Det Nom
3. Nom → Nom PP | N Nom | N
4. VP → VP PP | V NP | V NP PP | V PP
5. PP → P NP

N → flight | passenger | trip | morning | ...

V → is | prefers | like | need | depend | fly | ...

A → cheapest | non-stop | first | latest other | direct | ...

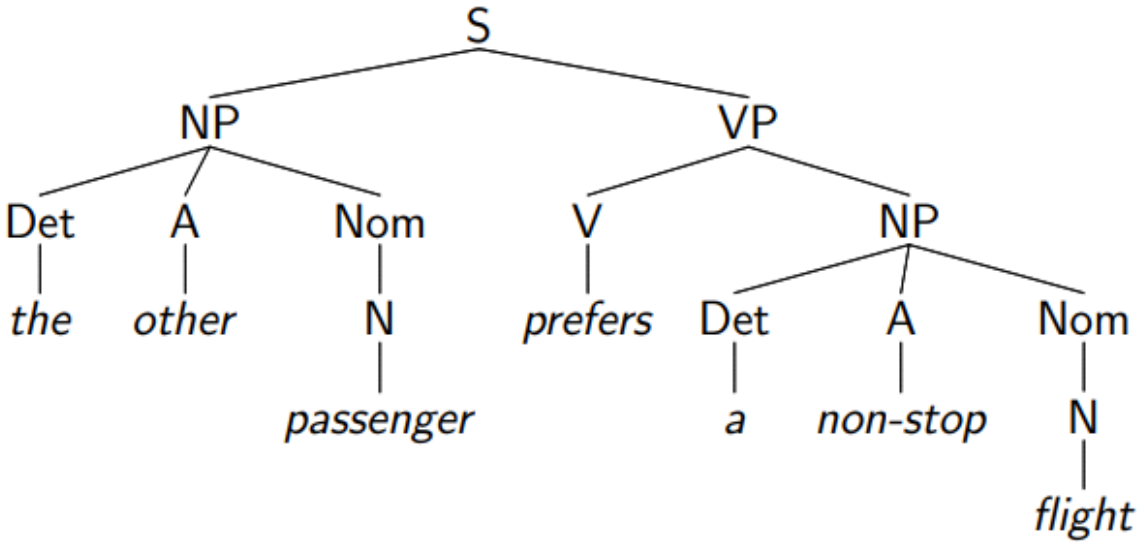
Pro → me | I | you | it | ...

PropN → Alaska | Baltimore | Los Angeles | Chicago | ...

Det → the | a | an | this | these | that | ...

P → from | to | on | near | ...

Примечание: грамматика реализует составление простейших предложений на английском языке.



Вариант 12

Для грамматики G[E] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[E]:

1. E → E “⇒” E1 | E1
2. E1 → E1 “∨” E2 | E2
3. E2 → E2 “∧” E3 | E3
4. E3 → “¬” E4 | E4
5. E4 →”(“ E “)” | id

id – идентификатор Б{Б|Ц}

Б – {a, b, c, ...z, A, B, …, Z}, Ц – {0, 1, …, 9}

Примечание: данными продукциями описывается язык булевой логики (в окне ввода текста замените символы эквивалентными по смыслу).

Вариант 13

Для грамматики G[HTML] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[HTML]:

1. HTML :== CONTENT | TAG HTML TAG
2. TAG :== “<html>” | “</html>” | “<head>” | “</head>” | “<title>” | “</title>” | “<body>” | “</body>” | “<h2>” | “</h2>” | “ ” | “ ”
3. CONTENT :== text

text – Б{Б}

Б – {a, b, c, ...z, A, B, …, Z}

Примечание: данными продукциями описывается язык HTML-разметки.

Вариант 14

Для грамматики G[STMT] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[STMT]:

1. STMT :== if “(” EXPR “)” STMT | if “(” EXPR “)” STMT else STMT
2. EXPR :== VARIABLE OPERATOR VALUE

VALUE – числовая константа Ц{Ц}, VARIABLE – переменная Б{Б|Ц}, Б – {a, b, c, ...z, A, B, …, Z}, Ц – {0, 1, …, 9}, OPERATOR – ”==” | ”<” | ”<=” | ”>” | ”>=” | ”!=”

Вариант 15

Для грамматики G[Doc] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[Doc]:

1. Doc -> λ | Element Doc
2. Element -> Text | Doc | Doc | List
3. List -> λ | ListItem List
4. ListItem -> Text
5. Text -> λ | Char Text

Char -> a | b | c | … | z | A | B | … | Z

Примечание: данными продукциями описывается язык HTML-разметки.

Вариант 16

Для грамматики G[Program] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[Program]:

1. Program -> ε | Instr Program
2. Instr -> '+' | '-' | '>' | '<' | ',' | '.' | '[' Program ']'

Примечание: данными продукциями описывается эзотерический язык программирования Brainfuck.

Вариант 17

Для грамматики G[Program] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[Program]:

1. Program -> ε | Instr Program
2. Instr -> '+' | '-' | '*' | '/' | '_' | '=' | '>' | '&' | '|' | '~' | '$' | '%' | '\' | '@' | '[' Program ']'

Примечание: данными продукциями описывается эзотерический язык программирования FALSE.

Вариант 18

Для грамматики G[S] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[S]:

1. S -> <Noun phrase> <Verb phrase>
2. <Noun phrase> -> <Noun>| <Adjective phrase> <Noun> | λ
3. <Verb phrase> -> <Verb><Noun phrase>
4. <Adjective phrase> -> <Adjective phrase ><Adjective> | λ

<Noun> -> flight | passenger | trip | morning | ...

<Verb> -> is | prefers | like | need | depend | fly | ...

<Adjective> -> non-stop | first | direct | ...

Вариант 19

Для грамматики G[stmt] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[stmt]:

1. stmt -> IF exp stmt | IF exp stmt ELSE stmt | ID ASSIGN exp SEMICOLON
2. exp -> TRUE | FALSE | exp OR exp | exp AND exp | NOT exp | exp

ID – переменная Б{Б|Ц}, Б – {a, b, c, ...z, A, B, …, Z}, Ц – {0, 1, …, 9}, ASSIGN – ”==” | ”<” | ”<=” | ”>” | ”>=” | ”!=”

Вариант 20

Для грамматики G[begin-stmt] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[begin-stmt]:

1. begin-stmt → begin stmt-list end
2. stmt-list → stmt | stmt ; stmt-list
3. stmt → if-stmt | while-stmt | begin-stmt | assg-stmt
4. if-stmt → if bool-expr then stmt else stmt
5. while-stmt → while bool-expr do stmt
6. assg-stmt → VAR := arith-expr
7. bool-expr → arith-expr compare-op arith-expr
8. arith-expr → VAR | NUM | (arith-expr) | arith-expr + arith-expr | arith-expr * arith-expr

VAR – переменная Б{Б|Ц}, Б – {a, b, c, ...z, A, B, …, Z}, NUM – {0, 1, …, 9}, compare-op – ”==” | ”<” | ”<=” | ”>” | ”>=” | ”!=”

Вариант 21

Для грамматики G[<AO>] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[<AO>]:

1. <AO> → <ID> = <AB>;
2. <AB> → T | T + < AB >
3. T → O | O * T
4. O → (- | ε)[< ID > | < FLOAT > | (<AB>)]
5. < ID > → Б{ Б | Ц }
6. <FLOAT> →Ч(.Ч | ε)

Ч → Ц { Ц } , Б → a | b |…| z | A | B | …| Z, Ц → 0 | 1 |…| 9

Примечание: данными продукциями представлена грамматика арифметического оператора присваивания для языка BASIC.

Вариант 22

Для грамматики G[expression] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[expression]:

1. expression := multExpression | powExpression | expression ('+'|'-') multExpression | expression ('+'|'-') powExpression
2. multExpression := ID | multExpression ('*'|'/') ID
3. powExpression := ID '^' powExpression | ID

ID – идентификатор Б{Б|Ц}, Б – {a, b, c, ...z, A, B, …, Z}, NUM – {0, 1, …, 9}

Вариант 23

Для грамматики G[begin-stmt] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[begin-stmt]:

1. begin-stmt → begin stmt-list end
2. stmt-list → stmt | stmt ; stmt-list
3. stmt → begin-stmt | assg-stmt
4. assg-stmt → VAR := arith-expr
5. arith-expr → VAR | NUM | (arith-expr) | arith-expr + arith-expr | arith-expr * arith-expr

VAR – переменная Б{Б|Ц}, Б – {a, b, c, ...z, A, B, …, Z}, NUM – {0, 1, …, 9}, compare-op – ”==” | ”<” | ”<=” | ”>” | ”>=” | ”!=”

Вариант 24

Для грамматики G[unsigned_number] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[unsigned_number]:

1. unsigned_number ::= decimal_number | exponential_part | decimal_number exponential_part
2. decimal_number ::= unsigned_integer | decimal_fraction | unsigned_integer decimal_fraction
3. unsigned_integer ::= digit | unsigned_integer digit
4. integer ::= unsigned_integer | '+' unsigned_integer | '-' unsigned_integer
5. decimal_fraction ::= '.' unsigned_integer
6. exponential_part ::= '10' integer

digit ::= 0 | 1 |…| 9

Примечание: данные продукции были взяты из грамматики языка ALGOl 60.

Вариант 25

Для грамматики G[<While>] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[<While>]:

1. <While> → while <Cond> do <Stmt> ;
2. <Cond>→ <LogExpr> {or <LogExpr>}
3. <LogExpr>→ <RelExpr> {and <RelExpr>}
4. <RelExpr>→<Operand> [rel <Operand>]
5. <Operand>→ var | const
6. <Stmt>→ var as <ArithExpr>
7. <ArithExpr>→<Operand> {ao <Operand>}

Примечание: while, do, and, or – ключевые слова. В тип rel объединили операции сравнения <,<=, >=, >, != и ==, в тип ao арифметические операции + и -, в тип as оператор присваивания =, тип var – название переменной (только буквы), тип const – числа. Причина, по которой не объединены в один тип логические операции and и or заключается в том, что эти операции имеют различный приоритет. Пример цепочки: while a < b and b <= c do b=b+c-20 end ;

Вариант 26

Для грамматики G[<While>] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[<While>]:

1. <While> → do <Stmt> while <Cond> ;
2. <Cond>→ <LogExpr> {or <LogExpr>}
3. <LogExpr>→ <RelExpr> {and <RelExpr>}
4. <RelExpr>→<Operand> [rel <Operand>]
5. <Operand>→ var | const
6. <Stmt>→ var as <ArithExpr>
7. <ArithExpr>→<Operand> {ao <Operand>}

Примечание: while, do, and, or – ключевые слова. В тип rel объединили операции сравнения <,<=, >=, >, != и ==, в тип ao арифметические операции + и -, в тип as оператор присваивания =, тип var – название переменной (только буквы), тип const – числа. Причина, по которой не объединены в один тип логические операции and и or заключается в том, что эти операции имеют различный приоритет. Пример цепочки: while a < b and b <= c do b=b+c-20 ;

Вариант 27

Для грамматики G[<For>] разработать и реализовать алгоритм анализа на основе метода рекурсивного спуска.

G[<For>]:

1. <For> → for id:=<Operand> to <Operand> do <Stmt> ;
2. <Operand>→ var | const
3. <Stmt>→ var as <ArithExpr>
4. <ArithExpr>→<Operand> {ao <Operand>}

Примечание: for, do, and, or – ключевые слова. В тип ao объединили арифметические операции + и -, в тип as оператор присваивания =, тип var – название переменной (только буквы), тип var – числа, тип id - идентификаторы (буквы и цифры). Причина, по которой не объединены в один тип логические операции and и or заключается в том, что эти операции имеют различный приоритет. Пример цепочки: for i3:=10 to n do x=y-z+70 ;