МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

**Лабораторная работа №5**
по дисциплине: Теория автоматов и формальных языков
тема: «Восходящая обработка контекстно-свободных языков методом “перенос-опознание”»

Выполнил: ст. группы ПВ-201
Морозов Данила Александрович

Проверил:
Рязанов Юрий Дмитриевич

Белгород 2022 г.

**Лабораторная работа №5**
**«Восходящая обработка контекстно-свободных языков методом “перенос-опознание”»**

**Цель работы:**

Изучить и научиться применять восходящие методы обработки формальных языков типа «перенос-опознание»

Задания к работе:

1. Преобразовать исходную КС-грамматику в грамматику без правой рекурсии (см. варианты заданий).

2. Определить множества ПЕРВЫХ для каждого символа грамматики.

3. Определить множества СЛЕДУЮЩИХ для каждого грамматики.

4. Построить управляющую таблицу восходящего МП-распознавателя типа «перенос-опознание». Если не удается построить управляющую таблицу МП-распознавателя без конфликтов типа «перенос-опознание», то преобразовать грамматику в грамматику, допускающую построение МП-распознавателя без конфликтов типа «перенос-опознание» и выполнить п.п. 2 — 4.5.

5. Построить процедуру опознания в виде конечного автомата.

6. Написать программу-распознаватель, реализующую построенный восходящий МП-распознаватель. После выполнения свертки программа должна выводить номер правила, по которому выполнилась свертка, и строку, состоящую из содержимого магазина и необработанной части входной цепочки.

7. Сформировать наборы тестовых данных. Тестовые данные должны содержать цепочки, принадлежащие языку, заданному грамматикой, (допустимые цепочки) и цепочки, не принадлежащие языку. Для каждой допустимой цепочки построить дерево вывода и правый вывод.

Каждое правило грамматики должно использоваться в выводах допустимых цепочек хотя бы один раз

8. Обработать цепочки из набора тестовых данных программой-распознавателем.

**Задание варианта:**

Вариант №9

1. $S \to S; O$
2. $S \to O;$
3. $O \to Y[S]$
4. $O \to Y[S][S]$
5. $O \to \{[S]Y\}$
6. $O \to \{Y[S]\}$
7. $O \to a = Y$
8. $Y \to (Y|Y)$
9. $Y \to (Y\&Y)$
10. $Y \to !\,(Y)$
11. $Y \to a$

**Выполнение работы:**

1. $S \to S; O$
2. $S \to O;$
3. $O \to Y[S]$
4. $O \to Y[S][S]$
5. $O \to \{[S]Y\}$
6. $O \to \{Y[S]\}$
7. $O \to a = Y$
8. $Y \to (Y|Y)$
9. $Y \to (Y\&Y)$
10. $Y \to !\,(Y)$
11. $Y \to a$

Определим множество ПЕРВЫХ.

| | **ПЕРВ** | СЛЕД |
|---|---|---|
| S | S O | |
| O | O Y { a | |
| Y | Y ( ! a | |

Добавим в ПЕРВ(O) значения ПЕРВ(Y). В ПЕРВ(S) значения ПЕРВ(O).

| | **ПЕРВ** | СЛЕД |
|---|---|---|
| S | S O Y { a ( ! | |
| O | O Y { a ( ! | |
| Y | Y ( ! a | |

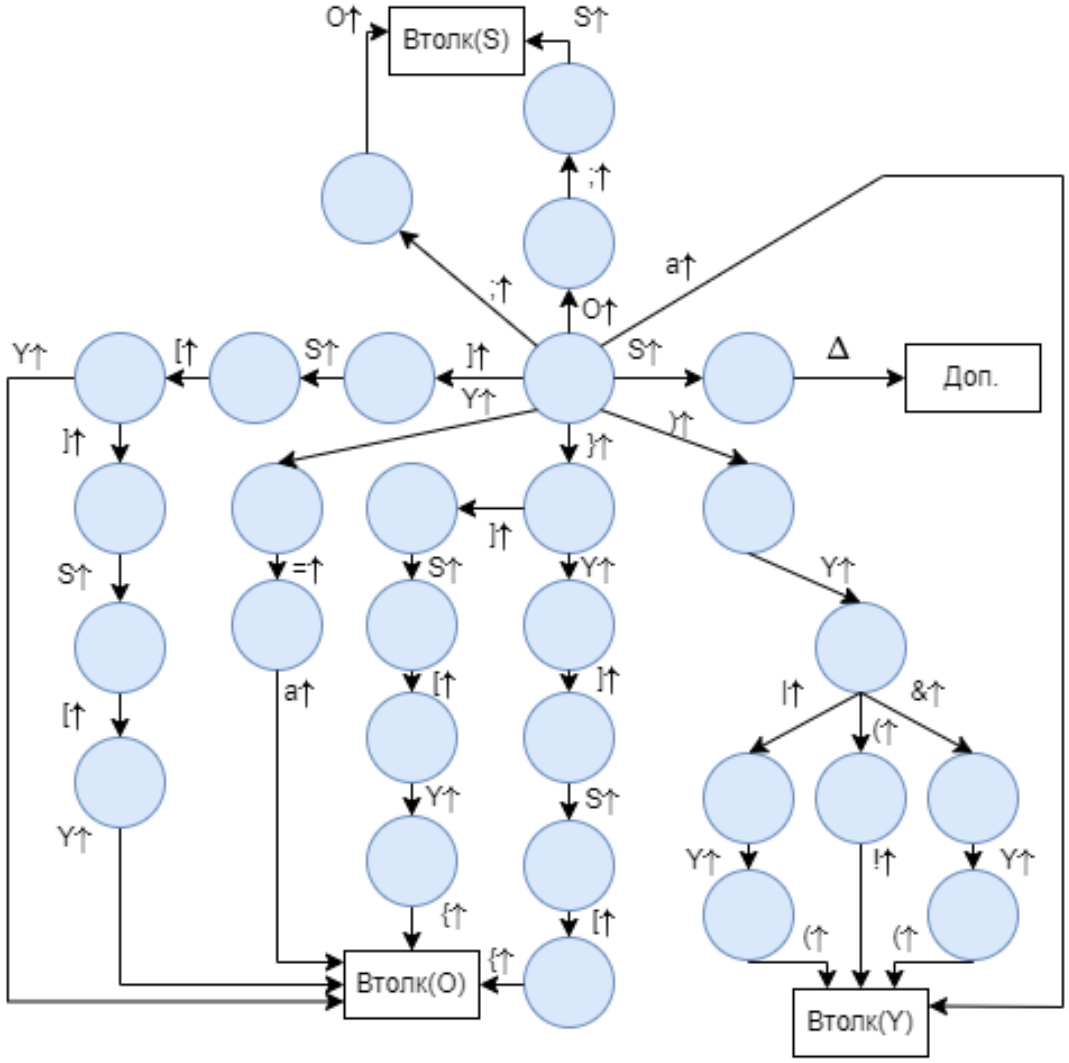
Найдем множества СЛЕД.

| | **ПЕРВ** | СЛЕД |
|---|---|---|
| S | S O Y { a ( ! | ⊣ ] ; |
| O | O Y { a ( ! | ; ⊣ ] |
| Y | Y ( ! a | [ } ) \| & ; ⊣ ] |

**Построим таблицу МП-распознавателя.**

| | ; | [ | ] | ( | ) | { | } | a | = | \| | & | ! | ⊣ | S | O | Y |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | П | | П | | | | | | | | | | ОП | | | |
| O | П | | ОП | | | | | | | | | | ОП | | | |
| Y | ОП | П | ОП | | П | | П | | | П | П | | ОП | | | |
| ; | ОП | | ОП | П | | П | | П | | | | П | ОП | | вт | вт |
| [ | | | | П | | П | | П | | | | П | | вт | вт | вт |
| ] | ОП | П | ОП | П | | | П | П | | | | П | ОП | | | |
| ( | | | | П | | | | П | | | | П | | | | вт |
| ) | ОП | ОП | ОП | | ОП | | ОП | | | ОП | ОП | | ОП | | | |
| { | | П | | П | | П | | П | | | | П | | | | вт |
| } | ОП | | ОП | | | | | | | | | | ОП | | | |
| a | ОП | ОП | ОП | | ОП | | ОП | | П | ОП | ОП | | ОП | | | |
| = | | | | П | | | | П | | | | П | | | | вт |
| \| | | | | П | | | | П | | | | П | | | | вт |
| & | | | | П | | | | П | | | | П | | | | вт |
| ! | | | | П | | | | | | | | | | | | |
| Δ | | | | П | | П | | П | | | | П | | вт | вт | вт |

Построим процедуру опознания в виде конечного автомата.



Напишем программу, реализующую вышеописанный МП-распознаватель.

mp.go

```
package mp

import (
    "fmt"
    "log"
    "strings"
)

const (
    ActionTransfer  = "ЗАМЕНИТЬ"
    ActionRecognize = "ОПОЗНАНИЕ"
    EndLineSymbol   = "⊣"
    EndStackSymbol  = "Δ"
)

type MPAction struct {
    Action string
}

var (
    MPRecognize MPAction = MPAction{Action: ActionRecognize}
    MPTransfer  MPAction = MPAction{Action: ActionTransfer}
)

var Table map[string]map[string]MPAction = map[string]map[string]MPAction{
    "S": {
        ";":          MPTransfer,
        "]":          MPTransfer,
        EndLineSymbol: MPRecognize,
    },
    "O": {
        ";":          MPTransfer,
        "]":          MPRecognize,
        EndLineSymbol: MPRecognize,
    },
    "Y": {
        ";":          MPRecognize,
        "[":          MPTransfer,
        "]":          MPRecognize,
        ")":          MPTransfer,
        "}":          MPTransfer,
        "|":          MPTransfer,
        "&":          MPTransfer,
        EndLineSymbol: MPRecognize,
    },
    ";": {
        ";":          MPRecognize,
```

```
        "]":           MPRecognize,
        "(":           MPTransfer,
        "{":           MPTransfer,
        "a":           MPTransfer,
        "!":           MPTransfer,
        EndLineSymbol: MPRecognize,
    },
    "[": {
        "(": MPTransfer,
        "{": MPTransfer,
        "a": MPTransfer,
        "!": MPTransfer,
    },
    "]": {
        ";":           MPRecognize,
        "]":           MPRecognize,
        "[":           MPTransfer,
        "}":           MPTransfer,
        "a":           MPTransfer,
        "(":           MPTransfer,
        "!":           MPTransfer,
        EndLineSymbol: MPRecognize,
    },
    "(": {
        "(": MPTransfer,
        "a": MPTransfer,
        "!": MPTransfer,
    },
    ")": {
        ";":           MPRecognize,
        "[":           MPRecognize,
        "]":           MPRecognize,
        ")":           MPRecognize,
        "}":           MPRecognize,
        "|":           MPRecognize,
        "&":           MPRecognize,
        EndLineSymbol: MPRecognize,
    },
    "{": {
        "[": MPTransfer,
        "(": MPTransfer,
        "{": MPTransfer,
        "a": MPTransfer,
        "!": MPTransfer,
    },
    "}": {
        ";":           MPRecognize,
        "]":           MPRecognize,
        EndLineSymbol: MPRecognize,
    },
    "a": {
```

```
        ";":           MPRecognize,
        "[":           MPRecognize,
        "]":           MPRecognize,
        ")":           MPRecognize,
        "}":           MPRecognize,
        "=":           MPTransfer,
        "|":           MPRecognize,
        "&":           MPRecognize,
        EndLineSymbol: MPRecognize,
    },
    "=": {
        "(": MPTransfer,
        "a": MPTransfer,
        "!": MPTransfer,
    },
    "|": {
        "(": MPTransfer,
        "a": MPTransfer,
        "!": MPTransfer,
    },
    "&": {
        "(": MPTransfer,
        "a": MPTransfer,
        "!": MPTransfer,
    },
    "!": {
        "(": MPTransfer,
    },
    EndStackSymbol: {
        "(": MPTransfer,
        "{": MPTransfer,
        "a": MPTransfer,
        "!": MPTransfer,
    },
}

var (
    Rules map[int][]string = map[int][]string{
        1:  {"S", "S;O"},
        2:  {"S", "O;"},
        3:  {"O", "Y[S]"},
        4:  {"O", "Y[S][S]"},
        5:  {"O", "{[S]Y}"},
        6:  {"O", "{Y[S]}"},
        7:  {"O", "a=Y"},
        8:  {"Y", "(Y|Y)"},
        9:  {"Y", "(Y&Y)"},
        10: {"Y", "!(Y)"},
        11: {"Y", "a"},
    }
)
```

```
type MP struct {
    stack      string
    Steps      []int
    CurrPos    int
    ScanLogger *log.Logger
}

func (mp *MP) Init(ScanLogger *log.Logger) {
    mp.stack = EndStackSymbol
    mp.ScanLogger = ScanLogger
}

func (mp *MP) Pop() error {
    runes := []rune(mp.stack)
    if mp.Pick() == EndStackSymbol {
        return fmt.Errorf("try of pop from the end of the stack")
    }
    mp.stack = string(runes[:len(runes)-1])
    return nil
}

type recognitionUnit struct {
    R int
    S string
}

var recognitionTable map[string]map[string]recognitionUnit =
map[string]map[string]recognitionUnit{
    "]": {
        "Y": {R: R3, S: "Y"},
        "]": {R: R4, S: "]S[Y"},
    },
    "}": {
        "]": {R: R6, S: "]S[Y{"},
        "Y": {R: R5, S: "Y]S[{"},
    },
    ")": {
        "|": {R: R8, S: "|Y("},
        "(": {R: R10, S: "(!"},
        "&": {R: R9, S: "&Y("},
    },
}

func reverse(s string) string {
    runes := []rune(s)
    for i, j := 0, len(runes)-1; i < j; i, j = i+1, j-1 {
        runes[i], runes[j] = runes[j], runes[i]
    }
    return string(runes)
}
```

```
func LastReplace(s string, old string, new string, n int) string {
    return reverse(strings.Replace(reverse(s), reverse(old), reverse(new), n))
}

func (mp *MP) Pick() string {
    return string([]rune(mp.stack)[len([]rune(mp.stack))-1])
}

func (mp *MP) TryReplace(s string, replacement string) error {
    for _, c := range s {
        if mp.Pick() == string(c) {
            mp.Pop()
        } else {
            return fmt.Errorf("wrong input")
        }
    }
    mp.stack = mp.stack + replacement
    return nil
}

func (mp *MP) TryDrop(s string) error {
    for _, c := range s {
        if mp.Pick() == string(c) {
            mp.Pop()
        } else {
            return fmt.Errorf("wrong input")
        }
    }
    return nil
}

const (
    R1 = iota + 1
    R2
    R3
    R4
    R5
    R6
    R7
    R8
    R9
    R10
    R11
    FINE
    DROP
)

func (mp *MP) Recognize() (int, error) {
    if mp.Pick() == "S" {
        mp.Pop()
```

```
        if mp.Pick() == EndStackSymbol {
            return FINE, nil
        } else {
            return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
        }
    }
    if mp.Pick() == "Y" {
        if err := mp.TryReplace("Y=a", "O"); err != nil {
            return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
        } else {
            return R7, nil
        }
    }
    if mp.Pick() == ";" {
        if err := mp.TryReplace(";O", "S"); err != nil {
            return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
        } else {
            return R2, nil
        }
    }
    if mp.Pick() == "O" {
        if err := mp.TryReplace("O;S", "S"); err != nil {
            return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
        } else {
            return R1, nil
        }
    }
    if mp.Pick() == "a" {
        if err := mp.TryReplace("a", "Y"); err != nil {
            return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
        } else {
            return R11, nil
        }
    }
    if mp.Pick() == ")" {
        if err := mp.TryDrop(")Y"); err != nil {
            return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
        }
        if v, ok := recognitionTable[")"][mp.Pick()]; !ok {
            return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
        } else {
            if err := mp.TryReplace(v.S, "Y"); err != nil {
                return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
            } else {
                return v.R, nil
            }
        }
    }
    if mp.Pick() == "]" {
        if err := mp.TryDrop("]S["); err != nil {
            return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
```

```
        }
        if v, ok := recognitionTable["]"][mp.Pick()]; !ok {
            return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
        } else {
            if err := mp.TryReplace(v.S, "O"); err != nil {
                return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
            } else {
                return v.R, nil
            }
        }
    }
    if mp.Pick() == "}" {
        if err := mp.TryDrop("}"); err != nil {
            return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
        }
        if v, ok := recognitionTable["}"][mp.Pick()]; !ok {
            return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
        } else {
            if err := mp.TryReplace(v.S, "O"); err != nil {
                return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
            } else {
                return v.R, nil
            }
        }
    }
    return DROP, fmt.Errorf("wrong stack input %s", mp.Pick())
}

func (mp *MP) Analyze(input string) error {
    str := strings.TrimSpace(input) + EndLineSymbol
    runes := []rune(str)
    process := true
    for process {
        stackSymbol := mp.Pick()
        strSymbol := string(runes[mp.CurrPos])
        action, ok := Table[stackSymbol][strSymbol]
        if !ok {
            mp.ScanLogger.Printf("\033[31mОшибка\033[0m\n")
            mp.ScanLogger.Printf("Stack: %s", mp.stack)
            mp.ScanLogger.Printf("Оставшаяся часть строки: %s\n", str[mp.CurrPos:])
            return fmt.Errorf("error during analyze. stack symbol <%s> and string
symbol <%s>", stackSymbol, strSymbol)
        }
        switch action.Action {
        case MPTransfer.Action:
            mp.stack = mp.stack + string([]rune(str)[mp.CurrPos])
            mp.CurrPos++
        case MPRecognize.Action:
            id, err := mp.Recognize()
            if err != nil {
                return err
```

```
            }
            if id != FINE {
                mp.Steps = append(mp.Steps, id)
                mp.ScanLogger.Printf("Применяется правило %d. \033[33m %s -> %s
\033[0m", id, Rules[id][0], Rules[id][1])
                mp.ScanLogger.Printf("Магазин:\t\033[34m%s\033[0m", mp.stack)
                mp.ScanLogger.Printf("Необработанная часть
цепочки:\t\033[34m%s\033[0m", string(runes[mp.CurrPos:]))
            } else {
                mp.ScanLogger.Printf("\033[32mЦепочка обработана успешно\033[0m")
                process = false
            }
        }
    }
    return nil
}
```

main.go

```
package main

import (
    "DanArmor/ta5/pkg/mp"
    "bufio"
    "fmt"
    "log"
    "os"
    "strings"
)

func main() {
    reader := bufio.NewReader(os.Stdin)
    var (
        str string
        err error
    )
    fmt.Print("Ввод: ")
    if str, err = reader.ReadString('\n'); err != nil {
        fmt.Println(err)
    }
    scannerLoggerSCAN := log.New(os.Stdout, "SCAN: ", log.Ltime)
    mpr := mp.MP{}
    mpr.Init(scannerLoggerSCAN)
    if err := mpr.Analyze(str); err != nil {
        fmt.Printf("\033[31mАнализ закончен с ошибкой: %s\033[0m", err.Error())
        return
    }
    steps := make([]int, len(mpr.Steps))
    copy(steps, mpr.Steps)
    for i, j := 0, len(steps)-1; i < j; i, j = i+1, j-1 {
```

```
        steps[i], steps[j] = steps[j], steps[i]
    }
    inner := "S"
    var output []string
    output = append(output, inner)
    for _, s := range steps {
        inner = mp.LastReplace(inner, mp.Rules[s][0], mp.Rules[s][1], 1)
        output = append(output, inner)
    }
    fmt.Printf("Правый вывод: %s\n", strings.Join(output, " \033[33m=>\033[0m "))
}
```

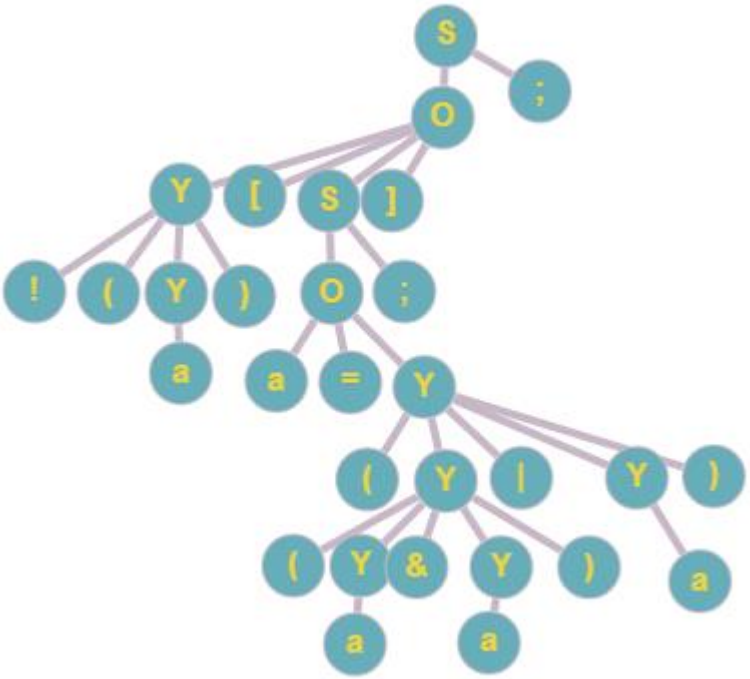
Сформируем тестовые данные. В задании сказано «Каждое правило грамматики должно использоваться в выводах допустимых цепочек хотя бы один раз.». Сформируем такие наборы тестовых данных, чтобы охватить все правила грамматики.

1. Допустимая цепочка: **!(a)[a=((a&a)|a);];**

   Порядок применения правил: 2, 3, 2, 7, 8, 11, 9, 11, 11, 10, 11

   Правый вывод: S ⇒ O; ⇒ Y[S]; ⇒ Y[O;]; ⇒ Y[a=Y;]; ⇒ Y[a=(Y|Y);]; ⇒ Y[a=(Y|a);]; ⇒ Y[a=((Y&Y)|a);]; ⇒ Y[a=((Y&a)|a);]; ⇒ Y[a=((a&a)|a);]; ⇒ !(Y)[a=((a&a)|a);]; ⇒ !(a)[a=((a&a)|a);];

   Дерево вывода:

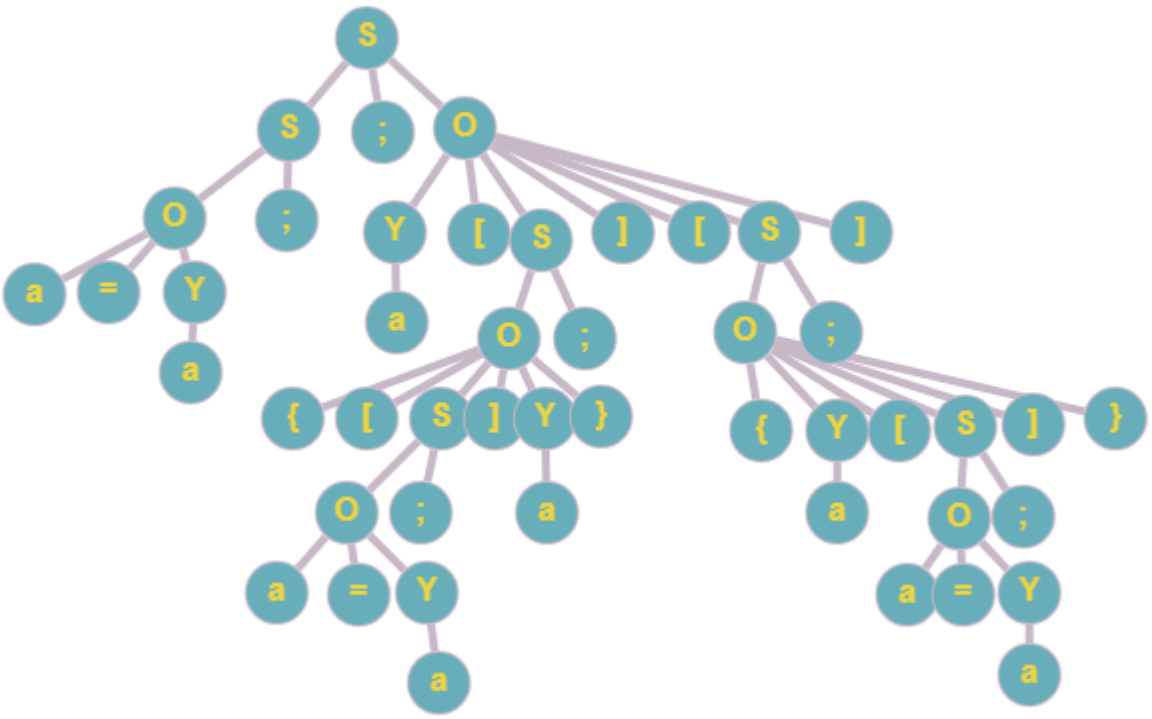


**2.** Допустимая цепочка: **a=a;;a[{[a=a;]a};][{a[a=a;]};]**

Порядок применения правил: 1 4 2 6 2 7 11 11 2 5 11 2 7 11 11 2 7 11

Правый вывод: S ⇒ S;O ⇒ S;Y[S][S] ⇒ S;Y[S][O;] ⇒ S;Y[S][{Y[S]};] ⇒ S;Y[S][{Y[O;]};] ⇒ S;Y[S][{Y[a=Y;]};] ⇒ S;Y[S][{Y[a=a;]};] ⇒ S;Y[S][{a[a=a;]};] ⇒ S;Y[O;][{a[a=a;]};] ⇒ S;Y[{[S]Y};][{a[a=a;]};] ⇒ S;Y[{[S]a};][{a[a=a;]};] ⇒ S;Y[{[O;]a};][{a[a=a;]};] ⇒ S;Y[{[a=Y;]a};][{a[a=a;]};] ⇒ S;Y[{[a=a;]a};][{a[a=a;]};] ⇒ S;a[{[a=a;]a};][{a[a=a;]};] ⇒ O;;a[{[a=a;]a};][{a[a=a;]};] ⇒ a=Y;;a[{[a=a;]a};][{a[a=a;]};] ⇒ a=a;;a[{[a=a;]a};][{a[a=a;]};]

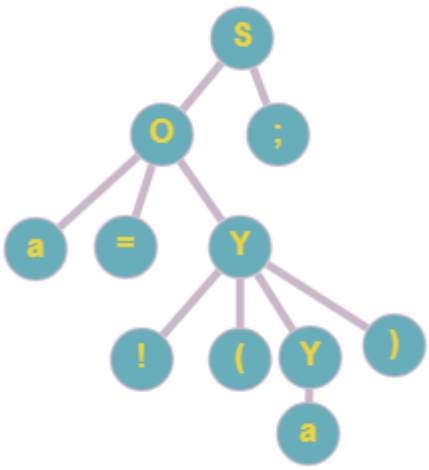
Дерево вывода:



3. Допустимая цепочка: **a=!(a);**
   Порядок применения правил: 2, 7, 10 ,11

   Правый вывод: S ⇒ O; ⇒ a=Y; ⇒ a=!(Y); ⇒ a=!(a);

   Дерево вывода:

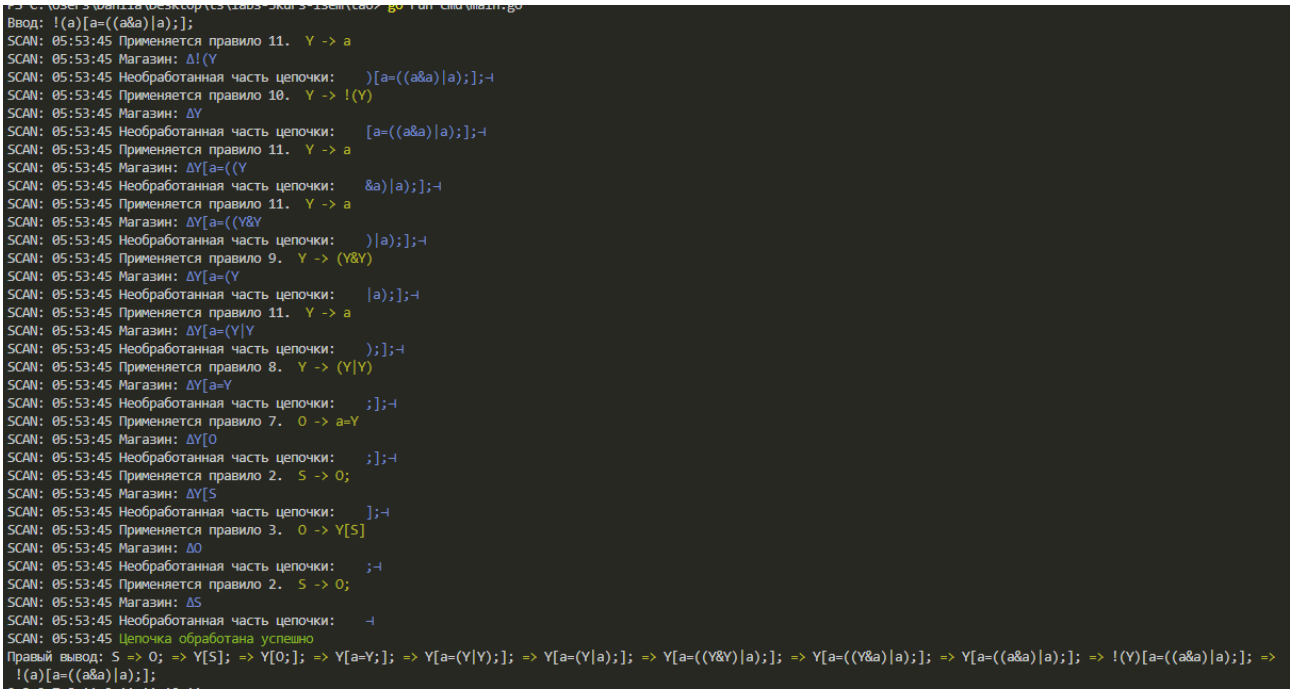



   Отметим, что три вышеописанных цепочки охватили все правила грамматики.

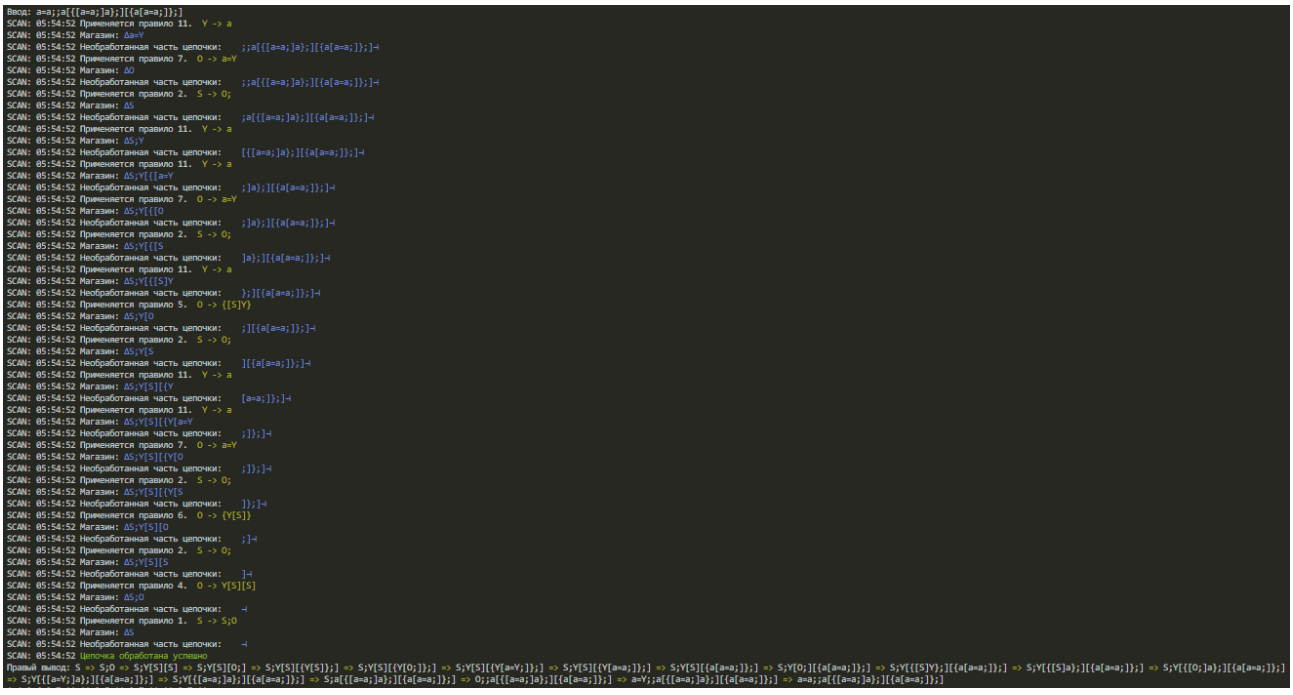
4. Недопустимая цепочка
   Пустая цепочка
5. Недопустимая цепочка
   a=a
6. Недопустимая цепочка
   a[a=a;;

Результаты работы реализованного МП-распознавателя:

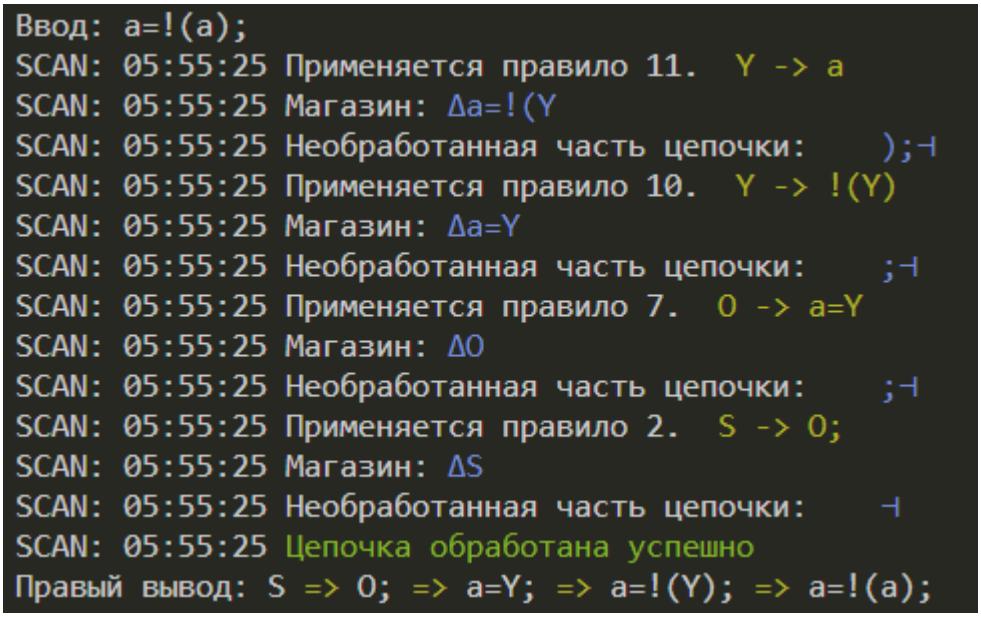
1.



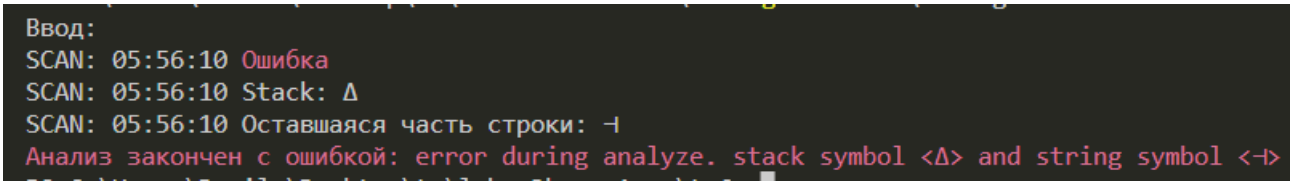

2.



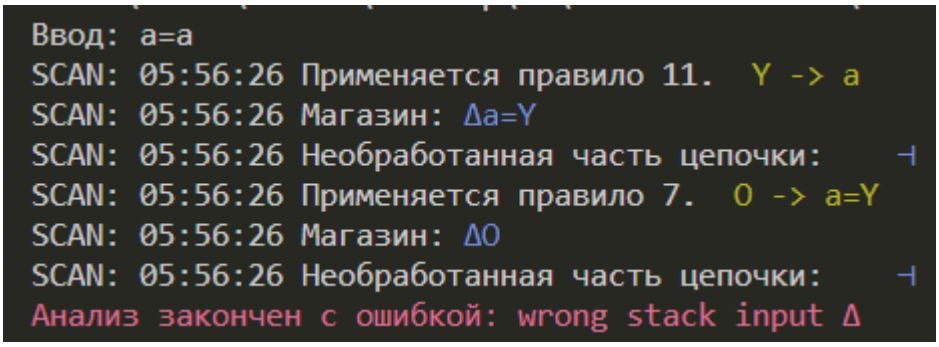
3.


Ввод: a=!(a);
SCAN: 05:55:25 Применяется правило 11. Y -> a
SCAN: 05:55:25 Магазин: Δa=!(Y
SCAN: 05:55:25 Необработанная часть цепочки: );⊣
SCAN: 05:55:25 Применяется правило 10. Y -> !(Y)
SCAN: 05:55:25 Магазин: Δa=Y
SCAN: 05:55:25 Необработанная часть цепочки: ;⊣
SCAN: 05:55:25 Применяется правило 7. O -> a=Y
SCAN: 05:55:25 Магазин: ΔO
SCAN: 05:55:25 Необработанная часть цепочки: ;⊣
SCAN: 05:55:25 Применяется правило 2. S -> O;
SCAN: 05:55:25 Магазин: ΔS
SCAN: 05:55:25 Необработанная часть цепочки: ⊣
SCAN: 05:55:25 Цепочка обработана успешно
Правый вывод: S => O; => a=Y; => a=!(Y); => a=!(a);

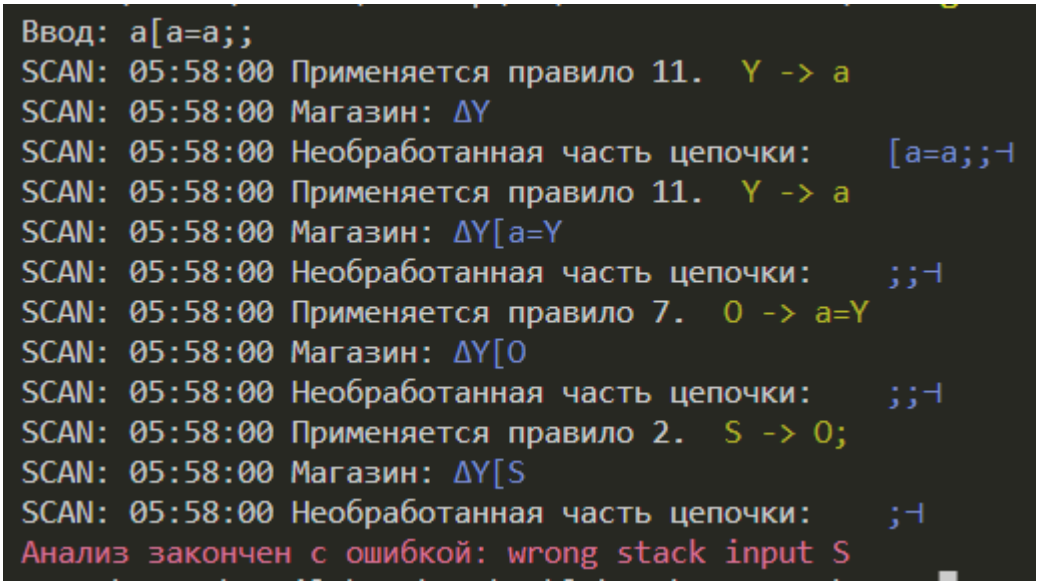
4.


Ввод:
SCAN: 05:56:10 Ошибка
SCAN: 05:56:10 Stack: Δ
SCAN: 05:56:10 Оставшаяся часть строки: ⊣
Анализ закончен с ошибкой: error during analyze. stack symbol <Δ> and string symbol <⊣>

5.


Ввод: a=a
SCAN: 05:56:26 Применяется правило 11. Y -> a
SCAN: 05:56:26 Магазин: Δa=Y
SCAN: 05:56:26 Необработанная часть цепочки: ⊣
SCAN: 05:56:26 Применяется правило 7. O -> a=Y
SCAN: 05:56:26 Магазин: ΔO
SCAN: 05:56:26 Необработанная часть цепочки: ⊣
Анализ закончен с ошибкой: wrong stack input Δ

6.




Все результаты совпали с ожидаемыми.

**Вывод:**

Мы изучили и научились применять восходящие методы обработки формальных языков типа «перенос-опознание», реализовав соответствующую программу.