

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 №4

по дисциплине: Теория автоматов и формальных языков
тема: «Нисходящая обработка контекстно-свободных языков»

Выполнил: ст. группы ПВ-201
Морозов Данила Александрович

Проверил:
Рязанов Юрий Дмитриевич

Белгород 2022 г.

Лабораторная работа №4

«Нисходящая обработка контекстно-свободных языков»

Цель работы:

Изучить и научиться применять нисходящие методы обработки формальных языков.

Задания к работе:

1. Преобразовать исходную КС-грамматику в LL(1)-грамматику (см. варианты заданий).
 2. Определить множества ПЕРВЫХ для каждого символа LL(1)-грамматики.
 3. Определить множества СЛЕДУЮЩИХ для каждого символа LL(1)-грамматики.
 4. Определить множество ВЫБОРА для каждого правила LL(1)-грамматики.
 5. Написать программу-распознаватель методом рекурсивного спуска. Программа должна выводить последовательность номеров правил, применяемых при левом выводе обрабатываемой цепочки.
 6. 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 левый вывод.
- Каждое правило грамматики должно использоваться в выводах допустимых цепочек хотя бы один раз.
7. Обработать цепочки из набора тестовых данных (см. п.6) программой-распознавателем.
 8. Построить нисходящий МП-распознаватель по LL(1)-грамматике.
 9. Написать программу-распознаватель, реализующую построенный нисходящий МП-распознаватель. Программа должна выводить на каждом шаге номер применяемого правила и промежуточную цепочку левого вывода.
 10. Обработать цепочки из набора тестовых данных (см. п.6) программой-распознавателем.

Задание варианта:

Вариант №9

1. $S \rightarrow OS$
2. $S \rightarrow O$
3. $O \rightarrow Y[S]$
4. $O \rightarrow \{Y[S]\}$
5. $O \rightarrow a = E$
6. $Y \rightarrow a = a$
7. $Y \rightarrow a < a$
8. $Y \rightarrow !(Y)$
9. $E \rightarrow (E + E)$
10. $E \rightarrow (E * E)$
11. $E \rightarrow -(E)$
12. $E \rightarrow a$

Выполнение работы:

1. $S \rightarrow OS$
2. $S \rightarrow O$
3. $O \rightarrow Y[S]$
4. $O \rightarrow \{Y[S]\}$
5. $O \rightarrow a = E$
6. $Y \rightarrow a = a$
7. $Y \rightarrow a < a$
8. $Y \rightarrow !(Y)$
9. $E \rightarrow (E + E)$
10. $E \rightarrow (E * E)$
11. $E \rightarrow -(E)$
12. $E \rightarrow a$

Преобразуем грамматику, выполнив левую факторизацию. Правило номер 3 выделено, потому что при раскрытии Y даст «а» в начале цепочки, что приведет к тому, что для правил 3 и 5 с одинаковой левой частью в начале правой части стоит один и тот же символ, что в конечном счете приведет к пересечению множеств ВЫБОРА для данных правил.

Подсветим правила, которые были изменены/добавлены.

1. $S \rightarrow OS'$
2. $S' \rightarrow OS'$
3. $S' \rightarrow \varepsilon$
4. $O \rightarrow aO'$
5. $O \rightarrow \{Y[S]\}$
6. $O \rightarrow !(Y)[S]$
7. $O' \rightarrow = E$
8. $O' \rightarrow Y'[S]$
9. $Y \rightarrow !(Y)$
10. $Y \rightarrow aY'$
11. $Y' \rightarrow = a$
12. $Y' \rightarrow < a$
13. $E \rightarrow -(E)$
14. $E \rightarrow a$
15. $E \rightarrow (EE'$
16. $E' \rightarrow * E)$
17. $E' \rightarrow +E)$

Но у нас появилась еще одна проблема: при раскрытии правила 8 мы вновь получим неоднозначность, ведь множество $\text{ВЫБОР}(O' \rightarrow Y'[S]) = \text{ПЕРВ}(Y') = \{< =\}$, что пересекается с $\text{ВЫБОР}(O' \rightarrow =E) = \{=\}$

Еще немного преобразуем грамматику.

1. $S \rightarrow OS'$
2. $S' \rightarrow OS'$
3. $S' \rightarrow \varepsilon$
4. $O \rightarrow aO'$
5. $O \rightarrow \{Y[S]\}$
6. $O \rightarrow !(Y)[S]$
7. $O' \rightarrow < a[S]$
8. $O' \rightarrow = R$
9. $R \rightarrow a[S]$
10. $R \rightarrow E$
11. $Y \rightarrow !(Y)$
12. $Y \rightarrow aY'$
13. $Y' \rightarrow = a$
14. $Y' \rightarrow < a$
15. $E \rightarrow -(E)$
16. $E \rightarrow a$
17. $E \rightarrow (EE'$
18. $E' \rightarrow * E)$
19. $E' \rightarrow +E)$

И хотелось бы уже начать определять множества, но мы опять попались в ловушку. Правила 9 и 10 имеет аналогичную проблему, которую мы решали на предыдущем шаге – множества ВЫБОРА этих двух правил пересекаются по символу «а».

Преобразуем грамматику

1. $S \rightarrow OS'$
2. $S' \rightarrow OS'$
3. $S' \rightarrow \varepsilon$
4. $O \rightarrow aO'$
5. $O \rightarrow \{Y[S]\}$
6. $O \rightarrow !(Y)[S]$
7. $O' \rightarrow < a[S]$
8. $O' \rightarrow = R$
9. $R \rightarrow -(E)$
10. $R \rightarrow (EE'$
11. $R \rightarrow aT$
12. $T \rightarrow [S]$
13. $T \rightarrow \varepsilon$
14. $Y \rightarrow !(Y)$
15. $Y \rightarrow aY'$
16. $Y' \rightarrow = a$
17. $Y' \rightarrow < a$

$$18. E \rightarrow -(E)$$

$$19. E \rightarrow a$$

$$20. E \rightarrow (EE'$$

$$21. E' \rightarrow * E)$$

$$22. E' \rightarrow +E)$$

Победа – правил, которые бы имели пересекающиеся множества ВЫБОРА не видно.

Теперь мы можем преступить к следующим заданиям, определять множества ПЕРВЫХ, СЛЕДУЮЩИХ и ВЫБОРА, где мы по итогу и увидим, что полученная грамматика является LL(1)-грамматикой.

Выполним поэтапно:

	Множества ПЕРВ	Множеств СЛЕД
S		
S'	ε	
O	$a \{ !$	
O'	$< =$	
R	$- (a$	
T	$\varepsilon [$	
Y	$a !$	
Y'	$= <$	
E	$a - ($	
E'	$+ *$	

В множество ПЕРВ(S') и ПЕРВ(S) добавим значения ПЕРВ(O) по 1-му и 2-му правилу.

	Множества ПЕРВ	Множеств СЛЕД
S	$a \{ !$	
S'	$\varepsilon a \{ !$	
O	$a \{ !$	
O'	$< =$	
R	$- (a$	
T	$\varepsilon [$	
Y	$a !$	
Y'	$= <$	
E	$a - ($	
E'	$+ *$	

При дальнейшем рассмотрении множества ПЕРВ не изменяются. Перейдем к следующему шагу.

Определим множества СЛЕД

	Множества ПЕРВ	Множеств СЛЕД
S	$a \{ !$	$\neg]$
S'	$\varepsilon a \{ !$	$\neg]$
O	$a \{ !$	$a \{ ! \neg]$
O'	$< =$	$a \{ ! \neg]$
R	$- (a$	$a \{ ! \neg]$

T	$\varepsilon [$	$a \{ ! \rightarrow]$
Y	$a !$	$[)$
Y'	$= <$	$[)$
E	$a - ($	$[) a \{ ! + *$
E'	$+ *$	$[) a \{ ! + *$

Мы определили множества СЛЕД.

Теперь определим множества ВЫБОРА для правил.

Грамматика	Множества ВЫБОР
1. $S \rightarrow OS'$	1. $a \{ !$
2. $S' \rightarrow OS'$	2. $a \{ !$
3. $S' \rightarrow \varepsilon$	3. $\rightarrow]$
4. $O \rightarrow aO'$	4. a
5. $O \rightarrow \{Y[S]\}$	5. $\{$
6. $O \rightarrow !(Y)[S]$	6. $!$
7. $O' \rightarrow < a[S]$	7. $<$
8. $O' \rightarrow = R$	8. $=$
9. $R \rightarrow -(E)$	9. $-$
10. $R \rightarrow (EE'$	10. $($
11. $R \rightarrow aT$	11. a
12. $T \rightarrow [S]$	12. $[$
13. $T \rightarrow \varepsilon$	13. $a \{ ! \rightarrow]$
14. $Y \rightarrow !(Y)$	14. $!$
15. $Y \rightarrow aY'$	15. a
16. $Y' \rightarrow = a$	16. $=$
17. $Y' \rightarrow < a$	17. $<$
18. $E \rightarrow -(E)$	18. $-$
19. $E \rightarrow a$	19. a
20. $E \rightarrow (EE'$	20. $($
21. $E' \rightarrow * E)$	21. $*$
22. $E' \rightarrow +E)$	22. $+$

У всех правил с одинаковой левой частью множества ВЫБОР попарно не пересекаются. Следовательно, мы имеем LL(1) грамматику.

Напишем программу-распознаватель методом рекурсивного спуска.

scan.go

```
package scan

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

type Scan struct {
    Str          string
    Runes        []rune
    CurrPos      int
    LoggerScan   log.Logger
    LoggerError  log.Logger
    TreeView     *Tree
    IsError      bool
    Error        error
}

type Tree struct {
    Children    []*Tree
    Label       string
    RuleNumber  int
    RuleString  string
}

func (s *Scan) nextSym() error {
    if s.CurrPos >= s.SizeRunes() {
        return fmt.Errorf("current pos is <%d> and length of string is <%d>",
s.CurrPos, s.SizeRunes())
    }
    s.CurrPos++
    return nil
}

func (s *Scan) SizeRunes() int {
    return len(s.Runes)
}

func (s *Scan) CurrRune() (rune, error) {
    if s.CurrPos >= s.SizeRunes() {
        return '\000', fmt.Errorf("current pos is <%d> and length of string is
<%d>", s.CurrPos, s.SizeRunes())
    }
    return s.Runes[s.CurrPos], nil
}
```

```

func (s *Scan) IsCurrEqual(r rune) error {
    if currRune, err := s.CurrRune(); currRune == r && err == nil {
        return nil
    } else {
        if err != nil {
            return fmt.Errorf("current rune <%c> is not equal to rune <%c>. %w",
currRune, r, err)
        } else {
            return fmt.Errorf("current rune <%c> is not equal to rune <%c>",
currRune, r)
        }
    }
}

func (s *Scan) PerfromStep(r rune) (node *Tree, err error) {
    if err = s.IsCurrEqual(r); err != nil {
        return node, err
    }
    node = &Tree{
        Label: string(r),
    }
    err = s.nextSym()
    return node, err
}

func (s *Scan) GenStep(r rune) func() (*Tree, error) {
    return func() (*Tree, error) {
        return s.PerfromStep(r)
    }
}

func (s *Scan) ThrowError(r rune) func() (*Tree, error) {
    return func() (*Tree, error) {
        return nil, fmt.Errorf("wrong input")
    }
}

func (s *Scan) Init(str string) error {
    s.Str = strings.TrimSpace(str)
    s.Runes = []rune(s.Str)
    s.IsError = false
    return nil
}

type (
    Step          func() (*Tree, error)
    SliceOfSteps []Step
)

func (s *Scan) RuleRecord(parent *Tree, number int, representation string) {

```

```

    s.LoggerScan.Printf("Применяется правило номер %d: \t\033[33m%s\033[0m\n", number, representation)
    parent.RuleNumber = number
    parent.RuleString = representation
}

func (s *Scan) ApplySteps(parent *Tree, steps SliceOfSteps) error {
    if s.IsError {
        return s.Error
    }
    var err error
    for _, fn := range steps {
        if node, err := fn(); err != nil {
            if node == nil {
                node = &Tree{
                    Label:      "ERROR",
                    RuleNumber: 0,
                }
            }
            s.Error = err
            s.IsError = true
            parent.Children = append(parent.Children, node)
            return err
        } else {
            parent.Children = append(parent.Children, node)
        }
    }
    return err
}

func (s *Scan) Analyze() error {
    s.LoggerScan.Printf("\033[32mНачало анализа\033[0m\n")
    parent, err := s.RuleS()
    s.TreeView = parent
    if err != nil {
        s.LoggerError.Printf("\033[31mАнализ закончен с ошибкой\033[0m\n")
        s.LoggerError.Printf("\033[31m%s\033[0m\n", err.Error())
    }
    return err
}

func (s *Scan) RuleS() (parent *Tree, err error) {
    parent = &Tree{
        Label: "S",
    }
    if s.CurrPos < s.SizeRunes() &&
        (s.IsCurrEqual('a') == nil || s.IsCurrEqual('{') == nil || s.IsCurrEqual('!') == nil) {
        s.RuleRecord(parent, 1, "S -> OS'")
        err = s.ApplySteps(parent, SliceOfSteps{s.Rule0, s.RuleS_})
    } else {

```

```

        s.RuleRecord(parent, 1, "S -> OS'")
        err = s.ApplySteps(parent, SliceOfSteps{s.ThrowError('a')}) // force
    }
    return parent, err
}

func (s *Scan) Epsilon() (parent *Tree, err error) {
    parent = &Tree{
        Label: "ε",
    }
    return parent, err
}

func (s *Scan) IsEnd() bool {
    return s.CurrPos >= s.SizeRunes()
}

func (s *Scan) RuleS_() (parent *Tree, err error) {
    parent = &Tree{
        Label: "S'",
    }
    if s.CurrPos < s.SizeRunes() &&
        (s.IsCurrEqual('a') == nil || s.IsCurrEqual('{') == nil || s.Is-
CurrEqual('!') == nil) {
        s.RuleRecord(parent, 2, "S' -> OS'")
        err = s.ApplySteps(parent, SliceOfSteps{s.Rule0, s.RuleS_})
    } else if s.IsEnd() || s.IsCurrEqual(']') == nil {
        s.RuleRecord(parent, 3, "S' -> ε")
        err = s.ApplySteps(parent, SliceOfSteps{s.Epsilon})
    } else {
        s.RuleRecord(parent, 2, "S' -> OS'")
        err = s.ApplySteps(parent, SliceOfSteps{s.ThrowError('a')}) // force
    }
    return parent, err
}

func (s *Scan) Rule0() (parent *Tree, err error) {
    parent = &Tree{
        Label: "0",
    }
    switch {
    case s.IsCurrEqual('a') == nil:
        s.RuleRecord(parent, 4, "0 -> a0'")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('a'), s.Rule0_})
    case s.IsCurrEqual('{') == nil:
        s.RuleRecord(parent, 5, "0 -> {Y[S]}")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('{'), s.RuleY, s.Gen-
Step('['), s.RuleS, s.GenStep(']'), s.GenStep('}')})
    case s.IsCurrEqual('!') == nil:
        s.RuleRecord(parent, 6, "0 -> !(Y)[S]")
    }
}

```

```

        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('!'), s.GenStep('('),
s.RuleY, s.GenStep(')'), s.GenStep('['), s.RuleS, s.GenStep(']')})
        default:
            s.RuleRecord(parent, 4, "O -> aO")
            err = s.ApplySteps(parent, SliceOfSteps{s.ThrowError('a')})
        }
        return parent, err
    }

func (s *Scan) RuleO_() (parent *Tree, err error) {
    parent = &Tree{
        Label: "O",
    }
    switch {
    case s.IsCurrEqual('<') == nil:
        s.RuleRecord(parent, 7, "O -> <a[S]")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('<'), s.GenStep('a'),
s.GenStep('['), s.RuleS, s.GenStep(']')})
    case s.IsCurrEqual('=') == nil:
        s.RuleRecord(parent, 8, "O -> =R")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('='), s.RuleR})
    default:
        s.RuleRecord(parent, 7, "O -> <a[S]")
        err = s.ApplySteps(parent, SliceOfSteps{s.ThrowError('<')})
    }
    return parent, err
}

func (s *Scan) RuleR() (parent *Tree, err error) {
    parent = &Tree{
        Label: "R",
    }
    switch {
    case s.IsCurrEqual('-') == nil:
        s.RuleRecord(parent, 9, "R -> -(E)")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('-'), s.GenStep('('),
s.RuleE, s.GenStep(')')})
    case s.IsCurrEqual('(') == nil:
        s.RuleRecord(parent, 10, "R -> (EE)")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('('), s.RuleE, s.RuleE_})
    case s.IsCurrEqual('a') == nil:
        s.RuleRecord(parent, 11, "R -> aT")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('a'), s.RuleT})
    default:
        s.RuleRecord(parent, 9, "O -> <a[S]")
        err = s.ApplySteps(parent, SliceOfSteps{s.ThrowError('-')})
    }
    return parent, err
}

func (s *Scan) RuleT() (parent *Tree, err error) {

```

```

    parent = &Tree{
        Label: "T",
    }
    switch {
    case s.IsCurrEqual '[' == nil:
        s.RuleRecord(parent, 12, "T -> [S]")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep '[' , s.RuleS, s.Gen-
Step(']')})
    case s.IsCurrEqual('a') == nil || s.IsCurrEqual('{') == nil ||
        s.IsCurrEqual('!') == nil || s.IsCurrEqual(']') == nil || s.IsEnd():
        s.RuleRecord(parent, 13, "T -> ε")
        err = s.ApplySteps(parent, SliceOfSteps{s.Epsilon})
    default:
        s.RuleRecord(parent, 12, "T -> [S]")
        err = s.ApplySteps(parent, SliceOfSteps{s.ThrowError '[' })
    }
    return parent, err
}

func (s *Scan) RuleY() (parent *Tree, err error) {
    parent = &Tree{
        Label: "Y",
    }
    switch{
    case s.IsCurrEqual('!') == nil:
        s.RuleRecord(parent, 14, "Y -> !(Y)")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('!'), s.GenStep('('),
s.RuleY, s.GenStep(')')})
    case s.IsCurrEqual('a') == nil:
        s.RuleRecord(parent, 15, "Y -> aY")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('a'), s.RuleY_})
    default:
        s.RuleRecord(parent, 14, "T -> [S]")
        err = s.ApplySteps(parent, SliceOfSteps{s.ThrowError('!')})
    }
    return parent, err
}

func (s *Scan) RuleY_() (parent *Tree, err error) {
    parent = &Tree{
        Label: "Y'",
    }
    switch{
    case s.IsCurrEqual('=') == nil:
        s.RuleRecord(parent, 16, "Y' -> =a")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('='), s.GenStep('a')})
    case s.IsCurrEqual('<') == nil:
        s.RuleRecord(parent, 17, "Y' -> <a")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('<'), s.GenStep('a')})
    default:
        s.RuleRecord(parent, 16, "Y' -> =a")

```

```

        err = s.ApplySteps(parent, SliceOfSteps{s.ThrowError('=')})
    }
    return parent, err
}

func (s *Scan) RuleE() (parent *Tree, err error) {
    parent = &Tree{
        Label: "E",
    }
    switch {
    case s.IsCurrEqual('(') == nil:
        s.RuleRecord(parent, 20, "E -> (EE'")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('('), s.RuleE, s.RuleE_})
    case s.IsCurrEqual('-') == nil:
        s.RuleRecord(parent, 18, "E -> -(E)")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('-'), s.GenStep('('),
s.RuleE, s.GenStep(')')})
    case s.IsCurrEqual('a') == nil:
        s.RuleRecord(parent, 19, "E -> a")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('a')})
    default:
        s.RuleRecord(parent, 19, "E -> a")
        err = s.ApplySteps(parent, SliceOfSteps{s.ThrowError('a')})
    }
    return parent, err
}

func (s *Scan) RuleE_() (parent *Tree, err error) {
    parent = &Tree{
        Label: "E'",
    }
    switch{
    case s.IsCurrEqual('*') == nil:
        s.RuleRecord(parent, 21, "E' -> *E'")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('*'), s.RuleE, s.Gen-
Step(')')})
    case s.IsCurrEqual('+') == nil:
        s.RuleRecord(parent, 22, "E' -> +E'")
        err = s.ApplySteps(parent, SliceOfSteps{s.GenStep('+'), s.RuleE, s.Gen-
Step(')')})
    default:
        s.RuleRecord(parent, 22, "E' -> +E'")
        err = s.ApplySteps(parent, SliceOfSteps{s.ThrowError('+')})
    }
    return parent, err
}

```

main.go

```
package main
```

```

import (
    "DanArmor/ta5/pkg/scan"
    "DanArmor/ta5/pkg/rules"
    "bufio"
    "fmt"
    "log"
    "os"
    "strings"
    "unicode"

    "github.com/m1gwings/treedrawer/tree"
)

type LeftOutput struct {
    inner string
    Steps []string
}

func (lo *LeftOutput) Init() {
    lo.inner = "S"
    lo.Steps = append(lo.Steps, "S")
}

func (lo *LeftOutput) Analyze(node *scan.Tree) {
    if unicode.IsUpper([]rune(node.Label)[0]){
        if(node.Label == "ERROR"){
            lo.inner = "ERROR"
            lo.Steps = append(lo.Steps, lo.inner)
        } else if node.RuleNumber != 0{
            lo.inner = strings.Replace(lo.inner, node.Label,
rules.Rules[node.RuleNumber], 1)
            lo.Steps = append(lo.Steps, lo.inner)
        }
        for _, c := range node.Children {
            lo.Analyze(c)
        }
    }
}

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)
    scannerLoggerERROR := log.New(os.Stderr, "ERROR: ", log.Ltime|log.Lshortfile)

```



```

s := scan.Scan{
    LoggerScan: *scannerLoggerSCAN,
    LoggerError: *scannerLoggerERROR,
}
if err = s.Init(str); err != nil {
    fmt.Println(err)
}
s.Analyze()
var drawFn func(node *scan.Tree, t *tree.Tree)
drawFn = func(node *scan.Tree, t *tree.Tree) {
    t.SetVal(tree.NodeString(node.Label))
    for _, c := range node.Children {
        tChild := t.AddChild(tree.NodeString(c.Label))
        drawFn(c, tChild)
    }
}
t := tree.NewTree(tree.NodeString(""))
drawFn(s.TreeView, t)
var lo LeftOutput
lo.Init()
lo.Analyze(s.TreeView)

fmt.Println(t)
strSteps := strings.Join(lo.Steps, "\033[33m => \033[0m")
fmt.Println("Левый вывод: ", strSteps)
}

```

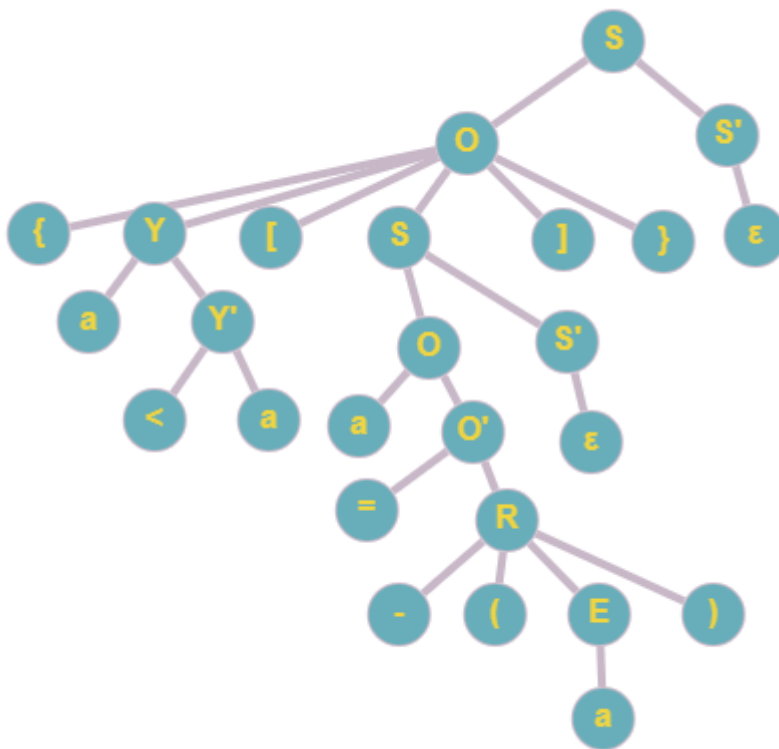
Сформируем тестовые данные. В задании сказано «Каждое правило грамматики должно использоваться в выводах допустимых цепочек хотя бы один раз.». Сформируем такие наборы тестовых данных, чтобы охватить все правила грамматики.

1. Допустимая цепочка: $\{a < a[a = -(a)]\}$

Порядок применения правил: 1, 5, 15, 17, 1, 4, 8, 9, 19, 3, 3

Левый вывод: $S \Rightarrow OS' \Rightarrow \{Y[S]\}S' \Rightarrow \{aY'[S]\}S' \Rightarrow \{a < a[S]\}S' \Rightarrow \{a < a[OS']\}S' \Rightarrow \{a < a[aO'S']\}S' \Rightarrow \{a < a[a=RS']\}S' \Rightarrow \{a < a[a=-(E)S']\}S' \Rightarrow \{a < a[a=-(a)S']\}S' \Rightarrow \{a < a[a=-(a)]\}S' \Rightarrow \{a < a[a=-(a)]\}$

Дерево вывода:

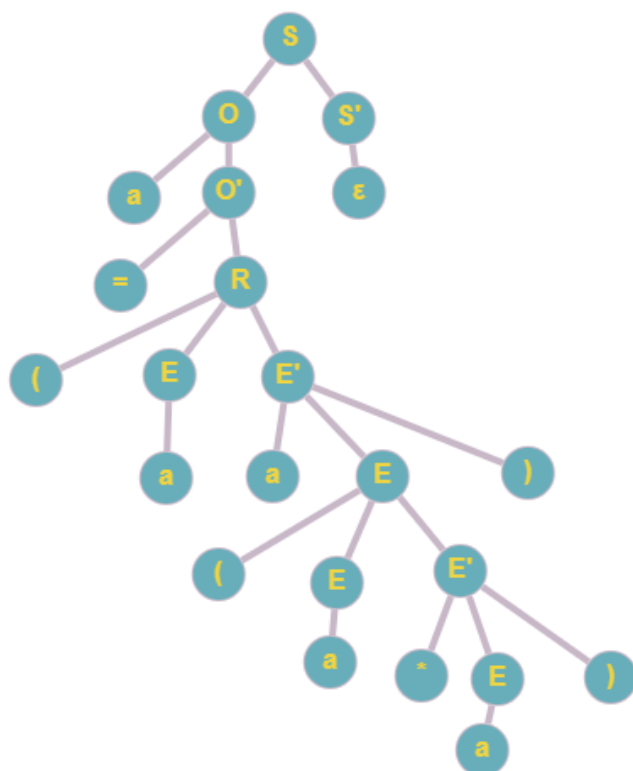


2. Допустимая цепочка: $a=(a+(a*a))$

Порядок применения правил: 1, 4, 8, 10, 19, 22, 20, 19, 21, 19, 3

Левый вывод: $S \Rightarrow OS' \Rightarrow aO'S' \Rightarrow a=RS' \Rightarrow a=(EE'S' \Rightarrow a=(aE'S' \Rightarrow a=(a+E)S' \Rightarrow a=(a+(EE')S' \Rightarrow a=(a+(aE')S' \Rightarrow a=(a+(a*E))S' \Rightarrow a=(a+(a*a))S' \Rightarrow a=(a+(a*a))$

Дерево вывода:



3. Допустимая цепочка: $!(!(a=a))[a=a[a=a]]a < a[a=(-(a)*a)]$

Порядок применения правил: 1, 6, 14, 15, 16, 1, 4, 8, 11, 12, 1, 4, 8, 11, 13, 3, 3, 2, 4, 7, 1, 4, 8, 10, 18, 19, 21, 19, 3, 3

Левый вывод: $S \Rightarrow OS' \Rightarrow !(Y)[S]S' \Rightarrow !(Y)[S]S' \Rightarrow !(aY)[S]S' \Rightarrow$
 $!(a=a)[S]S' \Rightarrow !(a=a)[OS']S' \Rightarrow !(a=a)[aO'S']S' \Rightarrow !(a=a)[a=RS']S' \Rightarrow$
 $!(a=a)[a=aTS']S' \Rightarrow !(a=a)[a=a[S]S']S' \Rightarrow !(a=a)[a=a[OS']S']S' \Rightarrow$
 $!(a=a)[a=a[aO'S']S']S' \Rightarrow !(a=a)[a=a[a=RS']S']S' \Rightarrow$
 $!(a=a)[a=a[a=aTS']S']S' \Rightarrow !(a=a)[a=a[a=aS']S']S' \Rightarrow$
 $!(a=a)[a=a[a=a]S']S' \Rightarrow !(a=a)[a=a[a=a]]S' \Rightarrow !(a=a)[a=a[a=a]]OS' \Rightarrow$
 $!(a=a)[a=a[a=a]]aO'S' \Rightarrow !(a=a)[a=a[a=a]]a < a[S]S' \Rightarrow$
 $!(a=a)[a=a[a=a]]a < a[OS']S' \Rightarrow !(a=a)[a=a[a=a]]a < a[aO'S']S' \Rightarrow$
 $!(a=a)[a=a[a=a]]a < a[a=RS']S' \Rightarrow !(a=a)[a=a[a=a]]a < a[a=(EE'S')]S' \Rightarrow$
 $!(a=a)[a=a[a=a]]a < a[a=(-(E)E'S')]S' \Rightarrow !(a=a)[a=a[a=a]]a < a[a=(-(a)E'S')]S'$
 $\Rightarrow !(a=a)[a=a[a=a]]a < a[a=(-(a)*E)S']S' \Rightarrow !(a=a)[a=a[a=a]]a < a[a=(-$
 $(a)*a)S']S' \Rightarrow !(a=a)[a=a[a=a]]a < a[a=(-(a)*a)]S' \Rightarrow$
 $!(a=a)[a=a[a=a]]a < a[a=(-(a)*a)]$

Дерево вывода:



Отметим, что три вышеописанных цепочки охватили все правила грамматики.

4. Недопустимая цепочка
Пустая цепочка
5. Недопустимая цепочка
 $a=(a+$
6. Недопустимая цепочка
 $\{a<a\}$

Результаты работы распознавателя, написанного методом рекурсивного спуска на тестовых данных:

1.

Ввод: {a<a[a--(a)]}

SCAN: 05:42:37 Начало анализа

SCAN: 05:42:37 Применяется правило номер 1: S -> OS'

SCAN: 05:42:37 Применяется правило номер 5: O -> {Y[S]}

SCAN: 05:42:37 Применяется правило номер 15: Y -> aY'

SCAN: 05:42:37 Применяется правило номер 17: Y' -> <a

SCAN: 05:42:37 Применяется правило номер 1: S -> OS'

SCAN: 05:42:37 Применяется правило номер 4: O -> aO'

SCAN: 05:42:37 Применяется правило номер 8: O' -> =R

SCAN: 05:42:37 Применяется правило номер 9: R -> -(E)

SCAN: 05:42:37 Применяется правило номер 19: E -> a

SCAN: 05:42:37 Применяется правило номер 3: S' -> ε

SCAN: 05:42:37 Применяется правило номер 3: S' -> ε

Левый вывод: S => OS' => {Y[S]}S' => {aY'[S]}S' => {a<a[S]}S' => {a<a[OS']}S' => {a<a[aO'S']}S' => {a<a[a=RS']}S' => {a<a[a--(E)S']}S' => {a<a[a--(a)S']}S' => {a<a[a--(a)]}S' => {a<a[a--(a)]}

2.

Ввод: $a=(a+(a*a))$

SCAN: 05:43:27 Начало анализа

SCAN: 05:43:27 Применяется правило номер 1: $S \rightarrow OS'$

SCAN: 05:43:27 Применяется правило номер 4: $O \rightarrow aO'$

SCAN: 05:43:27 Применяется правило номер 8: $O' \rightarrow =R$

SCAN: 05:43:27 Применяется правило номер 10: $R \rightarrow (EE'$

SCAN: 05:43:27 Применяется правило номер 19: $E \rightarrow a$

SCAN: 05:43:27 Применяется правило номер 22: $E' \rightarrow +E$

SCAN: 05:43:27 Применяется правило номер 20: $E \rightarrow (EE'$

SCAN: 05:43:27 Применяется правило номер 19: $E \rightarrow a$

SCAN: 05:43:27 Применяется правило номер 21: $E' \rightarrow *E$

SCAN: 05:43:27 Применяется правило номер 19: $E \rightarrow a$


SCAN: 05:43:27 Применяется правило номер 3: $S' \rightarrow \varepsilon$

Левый вывод: $S \Rightarrow OS' \Rightarrow aO'S' \Rightarrow a=RS' \Rightarrow a=(EE'S' \Rightarrow a=(aE'S' \Rightarrow a=(a+E)S' \Rightarrow a=(a+(EE')S' \Rightarrow a=(a+(aE'))S' \Rightarrow a=(a+(a*E))S' \Rightarrow a=(a+(a*a))S' \Rightarrow a=(a+(a*a))$

Вход:	1((([a-a])([a-a][a-a])[a-a-([a-a])])
SCAN:	05:43:55 Начало анализа
SCAN:	05:43:55 Применяется правило номер 1: S → OS'
SCAN:	05:43:55 Применяется правило номер 6: O → ([V])[S]
SCAN:	05:43:55 Применяется правило номер 14: Y → I(V)
SCAN:	05:43:55 Применяется правило номер 15: Y → aY'
SCAN:	05:43:55 Применяется правило номер 16: Y' → a
SCAN:	05:43:55 Применяется правило номер 1: S → OS'
SCAN:	05:43:55 Применяется правило номер 4: O → aO'
SCAN:	05:43:55 Применяется правило номер 8: O' → aR
SCAN:	05:43:55 Применяется правило номер 11: R → aT
SCAN:	05:43:55 Применяется правило номер 12: T → [S]
SCAN:	05:43:55 Применяется правило номер 1: S → OS'
SCAN:	05:43:55 Применяется правило номер 4: O → aO'
SCAN:	05:43:55 Применяется правило номер 8: O' → aR
SCAN:	05:43:55 Применяется правило номер 11: R → aT
SCAN:	05:43:55 Применяется правило номер 13: T → c
SCAN:	05:43:55 Применяется правило номер 3: S' → c
SCAN:	05:43:55 Применяется правило номер 3: S' → c
SCAN:	05:43:55 Применяется правило номер 2: S' → OS'
SCAN:	05:43:55 Применяется правило номер 4: O → aO'
SCAN:	05:43:55 Применяется правило номер 7: O' → c[a[S]
SCAN:	05:43:55 Применяется правило номер 1: S → OS'
SCAN:	05:43:55 Применяется правило номер 4: O → aO'
SCAN:	05:43:55 Применяется правило номер 8: O' → aR
SCAN:	05:43:55 Применяется правило номер 10: R → (EE'
SCAN:	05:43:55 Применяется правило номер 18: E → -(E'
SCAN:	05:43:55 Применяется правило номер 19: E → a
SCAN:	05:43:55 Применяется правило номер 21: E' → *E'
SCAN:	05:43:55 Применяется правило номер 19: E → a
SCAN:	05:43:55 Применяется правило номер 3: S' → c
SCAN:	05:43:55 Применяется правило номер 3: S' → c

[illegible]

```
PS C:\Users\Danila\Desktop\ts\labs-3kurs-1sem\ta5\cmd> go run .\main.go
Ввод:
SCAN: 05:50:16 Начало анализа
ERROR: 05:50:16 scan.go:128: Анализ закончен с ошибкой
ERROR: 05:50:16 scan.go:129: wrong input
```



```

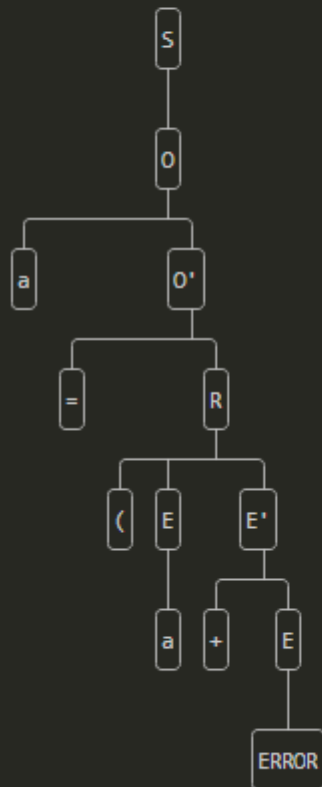
graph TD
    S[S] --> ERROR[ERROR]

```

Левый вывод: S => ERROR

5.

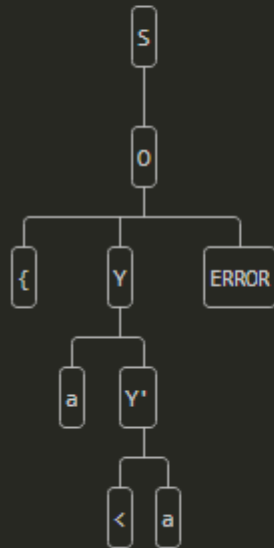
Ввод: a=(a+
 SCAN: 05:50:45 Начало анализа
 SCAN: 05:50:45 Применяется правило номер 1: $S \rightarrow OS'$
 SCAN: 05:50:45 Применяется правило номер 4: $O \rightarrow aO'$
 SCAN: 05:50:45 Применяется правило номер 8: $O' \rightarrow =R$
 SCAN: 05:50:45 Применяется правило номер 10: $R \rightarrow (EE'$
 SCAN: 05:50:45 Применяется правило номер 19: $E \rightarrow a$
 SCAN: 05:50:45 Применяется правило номер 22: $E' \rightarrow +E$
 ERROR: 05:50:45 scan.go:128: Анализ закончен с ошибкой
 ERROR: 05:50:45 scan.go:129: wrong input



Левый вывод: $S \Rightarrow OS' \Rightarrow aO'S' \Rightarrow a=RS' \Rightarrow a=(EE'S' \Rightarrow a=(aE'S' \Rightarrow a=(a+E)S' \Rightarrow \text{ERROR}$

6.

```
Ввод: {a<a]
SCAN: 05:51:07 Начало анализа
SCAN: 05:51:07 Применяется правило номер 1: S -> OS'
SCAN: 05:51:07 Применяется правило номер 5: O -> {Y[S]}
SCAN: 05:51:07 Применяется правило номер 15: Y -> aY'
SCAN: 05:51:07 Применяется правило номер 17: Y' -> <a
ERROR: 05:51:07 scan.go:128: Анализ закончен с ошибкой
ERROR: 05:51:07 scan.go:129: current rune <] is not equal to rune <[>
```



Левый вывод: $S \Rightarrow OS' \Rightarrow \{Y[S]\}S' \Rightarrow \{aY'[S]\}S' \Rightarrow \{a<a[S]\}S' \Rightarrow \text{ERROR}$

Все результаты совпали с ожидаемыми.

Построим таблицу МП-распознавателя.

	[]	{	}	a	=	<	!	()	*	+	-	¬
S			#1		#1			#1						
S'		#3	#2		#2			#2						#3
O			#5		#4			#6						
O'						#8	#7							
R					#11				#10				#9	
T	#12	#13	#13		#13			#13						#13
Y					#15			#14						
Y'						#16	#17							
E					#19				#20				#18	
E'											#21	#22		
]	ВЫТОЛК. сдвиг													
[ВЫТОЛК. сдвиг												
}				ВЫТОЛК. сдвиг										
a					ВЫТОЛК. сдвиг									
=						ВЫТОЛК. сдвиг								
(ВЫТОЛК. сдвиг					
)										ВЫТОЛК. сдвиг				
Δ														допустить

Н.с.м.: SΔ

#1. ЗАМЕНИТЬ(S'O), держать

#2. ЗАМЕНИТЬ(S'O), держать

#3. Вытолкнуть, держать

#4. ЗАМЕНИТЬ(O'), сдвиг

#5. ЗАМЕНИТЬ{ }S[Y), сдвиг

#6. ЗАМЕНИТЬ(JS()Y(), сдвиг

#7. ЗАМЕНИТЬ(JS[a), сдвиг

#8. ЗАМЕНИТЬ(R), сдвиг

#9. ЗАМЕНИТЬ(E), сдвиг

#10. ЗАМЕНИТЬ(E'E), сдвиг

#11. ЗАМЕНИТЬ(T), сдвиг

#12. ЗАМЕНИТЬ(JS), сдвиг

#13. Вытолкнуть, держать

#14. ЗАМЕНИТЬ()Y(), сдвиг

#15. ЗАМЕНИТЬ(Y'), сдвиг

#16. ЗАМЕНИТЬ(a), сдвиг

#17. ЗАМЕНИТЬ(a), сдвиг

#18. ЗАМЕНИТЬ()E), сдвиг
#19. ЗАМЕНИТЬ(), сдвиг
#20. ЗАМЕНИТЬ(E'E), сдвиг
#21. ЗАМЕНИТЬ()E), сдвиг
#22. ЗАМЕНИТЬ()E), сдвиг

Напишем программу, реализующую вышеописанный МП-распознаватель.

mp.go

```
package mp

import (
    "DanArmor/ta5/pkg/rules"
    "fmt"
    "log"
    "strings"
)

const (
    ActionReplace = "ЗАМЕНИТЬ"
    ActionPop     = "ВЫТОЛК"
    ActionDone    = "ДОПУСК"
    EndLineSymbol = "␣"
    EndStackSymbol = "Δ"
)

type MPAction struct {
    Move      bool
    Action    string
    RepalceString []string
    RuleNum   int
}

var (
    Rules []MPAction = []MPAction{
        {Move: false, Action: ActionReplace, RepalceString: []string{"S'", "O"},
        RuleNum: 1},
        {Move: false, Action: ActionReplace, RepalceString: []string{"S'", "O"},
        RuleNum: 2},
        {Move: false, Action: ActionPop, RuleNum: 3},
        {Move: true, Action: ActionReplace, RepalceString: []string{"O'"}, RuleNum:
4},
        {Move: true, Action: ActionReplace, RepalceString: []string{"", "]", "S",
"[", "Y"}, RuleNum: 5},
        {Move: true, Action: ActionReplace, RepalceString: []string{"", "S", "[",
")", "Y", "("}, RuleNum: 6},
        {Move: true, Action: ActionReplace, RepalceString: []string{"", "S", "[",
"a"}, RuleNum: 7},
```

```

        {Move: true, Action: ActionReplace, RepalceString: []string{"R"}, RuleNum:
8},
        {Move: true, Action: ActionReplace, RepalceString: []string{"", "E", "("},
RuleNum: 9},
        {Move: true, Action: ActionReplace, RepalceString: []string{"E", "E"},
RuleNum: 10},
        {Move: true, Action: ActionReplace, RepalceString: []string{"T"}, RuleNum:
11},
        {Move: true, Action: ActionReplace, RepalceString: []string{"", "S"},
RuleNum: 12},
        {Move: false, Action: ActionPop, RuleNum: 13},
        {Move: true, Action: ActionReplace, RepalceString: []string{"", "Y", "("},
RuleNum: 14},
        {Move: true, Action: ActionReplace, RepalceString: []string{"Y"}, RuleNum:
15},
        {Move: true, Action: ActionReplace, RepalceString: []string{"a"}, RuleNum:
16},
        {Move: true, Action: ActionReplace, RepalceString: []string{"a"}, RuleNum:
17},
        {Move: true, Action: ActionReplace, RepalceString: []string{"", "E", "("},
RuleNum: 18},
        {Move: true, Action: ActionReplace, RepalceString: []string{"", ""}, RuleNum:
19},
        {Move: true, Action: ActionReplace, RepalceString: []string{"E", "E"},
RuleNum: 20},
        {Move: true, Action: ActionReplace, RepalceString: []string{"", "E"},
RuleNum: 21},
        {Move: true, Action: ActionReplace, RepalceString: []string{"", "E"},
RuleNum: 22},
    }
    Rule_generic MPAction = MPAction{Move: true, Action: ActionPop}
    Rule_done    MPAction = MPAction{Move: false, Action: ActionDone}
)

var Table map[string]map[string]MPAction = map[string]map[string]MPAction{
    "S": {
        "{": Rules[0],
        "a": Rules[0],
        "!": Rules[0],
    },
    "S'": {
        "{": Rules[1],
        "]"": Rules[2],
        "a": Rules[1],
        "!": Rules[1],
        EndLineSymbol: Rules[2],
    },
    "O": {
        "{": Rules[4],
        "a": Rules[3],
        "!": Rules[5],
    },
}

```

```

    },
    "O'": {
        "=": Rules[7],
        "<": Rules[6],
    },
    "R": {
        "a": Rules[10],
        "(": Rules[9],
        "-": Rules[8],
    },
    "T": {
        "[": Rules[11],
        "]": Rules[12],
        "{": Rules[12],
        "a": Rules[12],
        "!": Rules[12],
        EndLineSymbol: Rules[12],
    },
    "Y": {
        "a": Rules[14],
        "!": Rules[13],
    },
    "Y'": {
        "=": Rules[15],
        "<": Rules[16],
    },
    "E": {
        "a": Rules[18],
        "(": Rules[19],
        "-": Rules[17],
    },
    "E'": {
        "*": Rules[20],
        "+": Rules[21],
    },
    "]:": {"]": Rule_generic},
    "[": {"[": Rule_generic},
    "}": {"}": Rule_generic},
    "a": {"a": Rule_generic},
    "=": {"=": Rule_generic},
    "(": {"(": Rule_generic},
    ")": {")": Rule_generic},
    EndStackSymbol: {EndLineSymbol: Rule_done},
}

func reverse(arr []string) {
    for i, j := 0, len(arr)-1; i < j; i, j = i+1, j-1 {
        arr[i], arr[j] = arr[j], arr[i]
    }
}

```

```

func init() {
    for i := 0; i < len(Rules); i++ {
        reverse(Rules[i].RepalceString)
    }
}

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

func (mp *MP) Init(ScanLogger *log.Logger) {
    mp.stack = append(mp.stack, "S")
    mp.stack = append(mp.stack, EndStackSymbol)
    mp.inner = "S"
    mp.ScanLogger = ScanLogger
}

func (mp *MP) Pop() error {
    if mp.stack[0] == EndStackSymbol {
        return fmt.Errorf("try of pop from the end of the stack")
    }
    mp.stack = mp.stack[1:]
    return nil
}

func (mp *MP) Replace(replacement []string) error {
    if mp.stack[0] == EndStackSymbol {
        return fmt.Errorf("try of replace the end of the stack")
    }
    mp.Pop()
    mp.stack = append(mp.stack, replacement...)
    copy(mp.stack[len(replacement):], mp.stack[:len(mp.stack)-len(replacement)])
    copy(mp.stack[:len(replacement)], replacement[:])
    if mp.stack[0] == "" {
        mp.Pop()
    }
    return nil
}

func (mp *MP) Analyze(input string) error {
    str := strings.TrimSpace(input) + EndLineSymbol
    runes := []rune(str)
    process := true
    mp.ScanLogger.Printf("Промежуточное представление:\t%s", mp.inner)
    for process {
        stackSymbol := mp.stack[0]
        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", strings.Join(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)
        }
        if action.Move {
            mp.CurrPos++
        }
        switch action.Action {
        case ActionDone:
            process = false
        case ActionPop:
            if err := mp.Pop(); err != nil {
                return err
            }
        case ActionReplace:
            if err := mp.Replace(action.RepalceString); err != nil {
                return err
            }
        }
        if action.RuleNum != 0 {
            if action.RuleNum == 3 || action.RuleNum == 13 {
                mp.ScanLogger.Printf("Применяется правило %d:\t\033[33m%s ->
%s\033[0m", action.RuleNum, stackSymbol, "ε")
            } else {
                mp.ScanLogger.Printf("Применяется правило %d:\t\033[33m%s ->
%s\033[0m", action.RuleNum, stackSymbol, rules.Rules[action.RuleNum])
            }
            mp.inner = strings.Replace(mp.inner, stackSymbol, rules.Rules[action.RuleNum], 1)
            mp.ScanLogger.Printf("Промежуточное представление:\t\033[34m%s\033[0m",
mp.inner)
        }
    }
    return nil
}

```

main.go

```

package main

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

```



```
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)
    }
    mp := mp.MP{}
    mp.Init(log.New(os.Stdout, "MP SCAN: ", log.Ltime))
    if err := mp.Analyze(str); err != nil {
        fmt.Printf("\033[31m%s\033[0m", err.Error())
    }
}
```

Результаты работы реализованного МП-распознавателя:

1.

```
Ввод: {a<a[a=-(a)]}
MP SCAN: 05:55:11 Промежуточное представление: S
MP SCAN: 05:55:11 Применяется правило 1: S -> OS'
MP SCAN: 05:55:11 Промежуточное представление: OS'
MP SCAN: 05:55:11 Применяется правило 5: O -> {Y[S]}
MP SCAN: 05:55:11 Промежуточное представление: {Y[S]}S'
MP SCAN: 05:55:11 Применяется правило 15: Y -> aY'
MP SCAN: 05:55:11 Промежуточное представление: {aY'[S]}S'
MP SCAN: 05:55:11 Применяется правило 17: Y' -> <a
MP SCAN: 05:55:11 Промежуточное представление: {a<a[S]}S'
MP SCAN: 05:55:11 Применяется правило 1: S -> OS'
MP SCAN: 05:55:11 Промежуточное представление: {a<a[OS']}S'
MP SCAN: 05:55:11 Применяется правило 4: O -> aO'
MP SCAN: 05:55:11 Промежуточное представление: {a<a[aO'S']}S'
MP SCAN: 05:55:11 Применяется правило 8: O' -> =R
MP SCAN: 05:55:11 Промежуточное представление: {a<a[a=RS']}S'
MP SCAN: 05:55:11 Применяется правило 9: R -> -(E)
MP SCAN: 05:55:11 Промежуточное представление: {a<a[a=-(E)S']}S'
MP SCAN: 05:55:11 Применяется правило 19: E -> a
MP SCAN: 05:55:11 Промежуточное представление: {a<a[a=-(a)S']}S'
MP SCAN: 05:55:11 Применяется правило 3: S' -> ε
MP SCAN: 05:55:11 Промежуточное представление: {a<a[a=-(a)]}S'
MP SCAN: 05:55:11 Применяется правило 3: S' -> ε
MP SCAN: 05:55:11 Промежуточное представление: {a<a[a=-(a)]}
```

2.

```
Ввод: a=(a+(a*a))
MP SCAN: 05:55:39 Промежуточное представление: S
MP SCAN: 05:55:39 Применяется правило 1: S -> OS'
MP SCAN: 05:55:39 Промежуточное представление: OS'
MP SCAN: 05:55:39 Применяется правило 4: O -> aO'
MP SCAN: 05:55:39 Промежуточное представление: aO'S'
MP SCAN: 05:55:39 Применяется правило 8: O' -> =R
MP SCAN: 05:55:39 Промежуточное представление: a=RS'
MP SCAN: 05:55:39 Применяется правило 10: R -> (EE'
MP SCAN: 05:55:39 Промежуточное представление: a=(EE'S'
MP SCAN: 05:55:39 Применяется правило 19: E -> a
MP SCAN: 05:55:39 Промежуточное представление: a=(aE'S'
MP SCAN: 05:55:39 Применяется правило 22: E' -> +E)
MP SCAN: 05:55:39 Промежуточное представление: a=(a+E)S'
MP SCAN: 05:55:39 Применяется правило 20: E -> (EE'
MP SCAN: 05:55:39 Промежуточное представление: a=(a+(EE')S'
MP SCAN: 05:55:39 Применяется правило 19: E -> a
MP SCAN: 05:55:39 Промежуточное представление: a=(a+(aE')S'
MP SCAN: 05:55:39 Применяется правило 21: E' -> *E)
MP SCAN: 05:55:39 Промежуточное представление: a=(a+(a*E))S'
MP SCAN: 05:55:39 Применяется правило 19: E -> a
MP SCAN: 05:55:39 Промежуточное представление: a=(a+(a*a))S'
MP SCAN: 05:55:39 Применяется правило 3: S' -> ε
MP SCAN: 05:55:39 Промежуточное представление: a=(a+(a*a))
```

3.

[illegible]

4.

```
Ввод:
MP SCAN: 05:56:37 Промежуточное представление: S
MP SCAN: 05:56:37 Ошибка
MP SCAN: 05:56:37 Stack: SΔ
MP SCAN: 05:56:37 Оставшаяся часть строки: -\
error during analyze. stack symbol <S> and string symbol <->
```

5.

```
Ввод: a=(a+
MP SCAN: 05:56:58 Промежуточное представление: S
MP SCAN: 05:56:58 Применяется правило 1: S -> OS'
MP SCAN: 05:56:58 Промежуточное представление: OS'
MP SCAN: 05:56:58 Применяется правило 4: O -> aO'
MP SCAN: 05:56:58 Промежуточное представление: aO'S'
MP SCAN: 05:56:58 Применяется правило 8: O' -> =R
MP SCAN: 05:56:58 Промежуточное представление: a=RS'
MP SCAN: 05:56:58 Применяется правило 10: R -> (EE'S'
MP SCAN: 05:56:58 Промежуточное представление: a=(EE'S'
MP SCAN: 05:56:58 Применяется правило 19: E -> a
MP SCAN: 05:56:58 Промежуточное представление: a=(aE'S'
MP SCAN: 05:56:58 Применяется правило 22: E' -> +E)
MP SCAN: 05:56:58 Промежуточное представление: a=(a+E)S'
MP SCAN: 05:56:58 Ошибка
MP SCAN: 05:56:58 Stack: E)S'Δ
MP SCAN: 05:56:58 Оставшаяся часть строки: -\
error during analyze. stack symbol <E> and string symbol <->
```

6.

```
Ввод: {a<a}
MP SCAN: 05:57:13 Промежуточное представление: S
MP SCAN: 05:57:13 Применяется правило 1: S -> OS'
MP SCAN: 05:57:13 Промежуточное представление: OS'
MP SCAN: 05:57:13 Применяется правило 5: O -> {Y[S]}
MP SCAN: 05:57:13 Промежуточное представление: {Y[S]}S'
MP SCAN: 05:57:13 Применяется правило 15: Y -> aY'
MP SCAN: 05:57:13 Промежуточное представление: {aY'[S]}S'
MP SCAN: 05:57:13 Применяется правило 17: Y' -> <a
MP SCAN: 05:57:13 Промежуточное представление: {a<a[S]}S'
MP SCAN: 05:57:13 Ошибка
MP SCAN: 05:57:13 Stack: [S]}S'Δ
MP SCAN: 05:57:13 Оставшаяся часть строки: ]-\
error during analyze. stack symbol <[> and string symbol <]>
```

Все результаты совпали с ожидаемыми и полученными предыдущей программой.

Вывод:

Мы изучили и научились применять нисходящие методы обработки формальных языков, реализовав распознаватель методом рекурсивного спуска и МП-распознаватель.