

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА  
ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Отчёт по дисциплине «Математическая логика»

Лабораторная работа №5  
«Множества **FIRST** и **FOLLOW**»  
Вариант №14

Студент: \_\_\_\_\_

Салимли Айзек Мухтар Оглы

Преподаватель: \_\_\_\_\_

Востров Алексей Владимирович

«\_\_\_\_» \_\_\_\_\_ 20\_\_ г.

Санкт-Петербург, 2025

# Содержание

Введение	3
1 Математическое описание	4
2 LL(k)-грамматики	5
3 Множества FIRST и FOLLOW	6
4 Левый анализатор LL(1)	7
5 Семантические действия	9
6 Программная реализация	12
7 Результаты программы	22
Заключение	25
Список литературы	26

## Введение

Задана грамматика:

1.  $S' \rightarrow S\$$ ,
2.  $S \rightarrow AaS|b$ ,
3.  $A \rightarrow CAb|B$ ,
4.  $B \rightarrow cSa|\epsilon$ ,
5.  $C \rightarrow c|ab$

Необходимо:

- Построить множества FIRST и FOLLOW для каждого нетерминала грамматики и таблицу выбора (lookup table).
- Реализовать детерминированный левый анализатор (проверка принадлежности цепочки грамматике).
- Назначить семантические действия части заданных продукций.

В качестве семантических действий части заданных продукций был выбран вывод функции `printStrLn` "вывод цепочки на Haskell".

# 1 Математическое описание

Контекстно-свободные грамматики задаются продукциями следующего вида  $A \rightarrow \beta$ , где  $A$  — нетерминал,  $\beta$  — произвольная цепочка из терминалов и нетерминалов.

Контекстно-свободной грамматикой называется грамматика, у которой левые части всех продукций являются одиночными нетерминалами. Контекстно-свободные грамматики являются грамматиками Хомского типа 2.

Заданная грамматика:

Исходная грамматика	Итоговая (LL(1)) грамматика
$S' \rightarrow S\$$	$S' \rightarrow S\$$
$S \rightarrow A a S \mid b$	$S \rightarrow A a S \mid b$
$A \rightarrow C A b \mid B$	$A \rightarrow C A b \mid B$
$B \rightarrow c S a \mid \epsilon$	$B \rightarrow c S a \mid \epsilon$
$C \rightarrow c \mid ab$	$C \rightarrow a \mid b$

**Причина изменения:** В исходной грамматике альтернативы  $A \rightarrow C A b$  и  $A \rightarrow B$  конфликтовали при LL(1)-анализе: при выводе нетерминала  $C$  в терминал  $c$  обе альтернативы имели одинаковый префикс  $\langle c, \dots \rangle$ . Из-за этого в таблице разбора возникало пересечение  $\text{FIRST}(C A b) \cap \text{FIRST}(B) \neq \emptyset$ .

**Сделанное преобразование:** Изменено правило  $C \rightarrow c \mid ab$  на  $C \rightarrow a \mid b$ . После этого

$$\text{FIRST}(C) = \{a, b\}, \quad \text{FIRST}(B) = \{\epsilon, c\},$$

и пересечение с  $\text{FIRST}(C A b) = \{a, b\}$  стало пустым. Тем самым грамматика удовлетворяет условию LL(1), и предиктивный парсер строится без конфликтов.

Преобразование не увеличило мощность описываемого языка (исключены только строки, начинающиеся одиночным терминалом  $c$ ), но позволило использовать детерминированный LL(1)-анализатор без возврата и ручного разрешения конфликтов.

## 2 LL( $k$ )-грамматики

LL( $k$ )-грамматики — это наиболее общий класс грамматик, позволяющих выполнить нисходящий синтаксический анализ, просматривая входную цепочку слева при восстановлении левого канонического вывода данной терминальной цепочки, заглядывая вперёд по входной цепочке на каждом шаге не более, чем на  $k$  символов при принятии решения о том, какой из альтернативных правых частей заменить текущий — самый левый — нетерминал очередной сентенциальной формы.

Промежуточная цепочка в процессе вывода состоит из цепочки терминалов  $wu$ , самого левого нетерминала  $A$  и недоведённой части  $x$ .

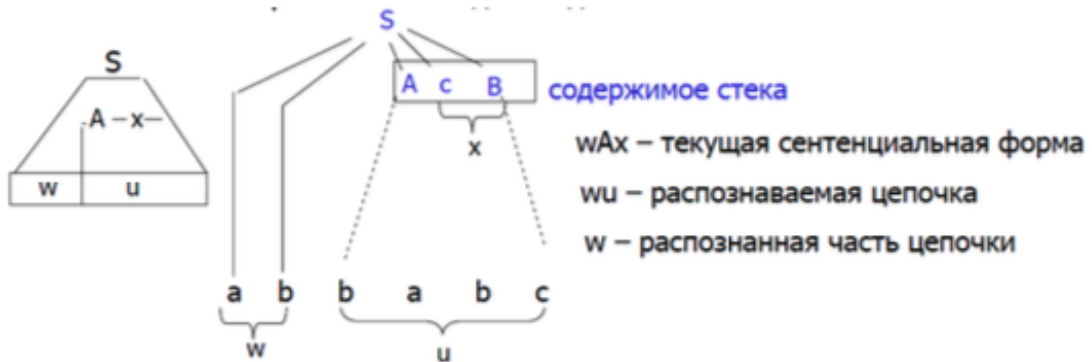


Рис. 1: LL( $k$ )-грамматика

### 3 Множества FIRST и FOLLOW

Нетерминал	FIRST	FOLLOW
$S'$	$\{a, b, c\}$	$\{\$ \}$
$S$	$\{a, b, c\}$	$\{a, \$ \}$
$A$	$\{a, b, c, \epsilon\}$	$\{a, b\}$
$B$	$\{c, \epsilon\}$	$\{a, b\}$
$C$	$\{a, b\}$	$\{a, b\}$

#### Замечания.

- Терминал «\$» — служебный символ конца ввода.
- Пустое слово обозначено как  $\epsilon$ .
- Для  $A$  и  $B$  в FIRST присутствует  $\epsilon$ , так как правила  $A \rightarrow B$  и  $B \rightarrow \epsilon$  допускают порождение пустой цепочки.
- В FOLLOW( $C$ ) попадают  $\{a, b\}$ , так как после  $C$  в правиле  $A \rightarrow CA b$  стоит нетерминал  $A$  (дать его  $\text{FIRST} \setminus \{\epsilon\}$  и терминал  $b$ ; а из-за наличия  $\epsilon \in \text{FIRST}(A)$  добавляется и всё FOLLOW( $A$ )).

Наша обновлённая грамматика — LL(1), потому что для каждого нетерминала достаточно одного символа look-ahead, чтобы однозначно выбрать альтернативу. Для любого нетерминала его альтернативы имеют непересекающиеся множества FIRST. Если у некоторой альтернативы в FIRST находится  $\epsilon$ , то она также непересекается с FOLLOW того же нетерминала. Тем самым предиктивное множество ( $\text{FIRST} \cup \text{FOLLOW}$ , когда применимо) каждой альтернативы уникально — конфликтов нет.

Эквивалентно этому, при построении «таблицы выбора» (lookup-table) каждая ячейка (нетерминал, символ look-ahead) заполняется не более чем одним правилом; таблица не содержит пересечений — что и подтверждает свойство LL(1).

Нетерм.	look-ahead	применяемое правило
$S$	$a$	$S \rightarrow A a S$
$S$	$b$	$S \rightarrow b$
$S$	$c$	$S \rightarrow A a S$
$A$	$a$	$A \rightarrow C A b$
$A$	$b$	$A \rightarrow C A b$
$A$	$c$	$A \rightarrow B$
$A$	$(\epsilon)/b$	$A \rightarrow B$
$B$	$c$	$B \rightarrow c S a$
$B$	$a, b, \$$	$B \rightarrow \epsilon$
$C$	$a$	$C \rightarrow a$
$C$	$b$	$C \rightarrow b$

## 4 Левый анализатор LL(1)

Так как заданная грамматика является LL(1), то для её анализа будет использован LL(1)-анализатор. LL(1) анализаторы, которые просматривают поток только на один символ вперед при принятии решения о том, какое правило грамматики необходимо применить.

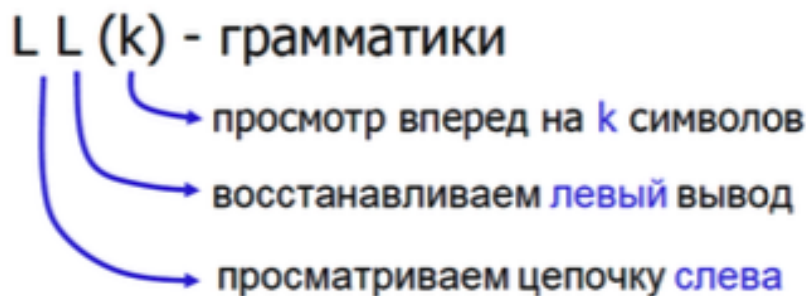


Рис. 2: LL(1)-анализатор

Реализован классический анализатор-предиктор с магазинной памятью. Его стек служит «планом» ещё не выполненной части вывода; элементами стека являются все терминальные и нетерминальные символы грамматики, а также служебный символ \$, обозначающий конец ввода.

### 1. Инициализация

В стек заносится пара  $S' \$$  (снизу — \$, сверху — начальный нетерминал  $S'$ ). Указатель входа ставится на первый терминал обрабатываемой цепочки.

### 2. Основной цикл

Пока стек не пуст, рассматриваем символ на его вершине и текущий входной терминал.

Ситуация на вершине стека	Действие анализатора
Терминал совпадает с текущим символом входа	<i>match</i> : снимаем терминал со стека, считываем следующий входной символ
Терминал не совпадает	<i>ошибка</i> : входная цепочка не соответствует грамматике
Нетерминал $A$	по паре $(A, a_1)$ смотрим look-up таблицу и находим единственную подходящую продукцию $A \rightarrow \beta$ ; заменяем $A$ на $\beta$ (символы $\beta$ кладутся в стек в обратном порядке, чтобы первый символ $\beta$ оказался сверху)
Для пары $(A, a_1)$ нет правила	<i>ошибка</i> (недопустимый вход)

При замене нетерминала сразу выполняется присвоенное ему **семантическое действие**: из сгенерированных потомками фраз собирается фрагмент Haskell-кода (`putStrLn "ab", "c"++ ...`, и т. д.).

### 3. Завершение

Анализ оканчивается успехом, если одновременно выполнены условия:

- стек пуст (всё «расписание вывода» отработано);
- входная цепочка исчерпана и последний считанный символ — \$.

В этом случае накопленный семантический атрибут корня ( $S'$ ) возвращается как результирующий Haskell-код.

### 4. Обнаружение ошибок

Ошибка фиксируется в двух случаях:

- терминал на вершине стека не совпал с текущим входным символом;

- пара (A, a) на вершине не соответствует ни одна запись таблицы выбора для данного look-ahead.

В программе выводится диагностическое сообщение вида:

Expected 'a', saw 'c' или No rule for A on 'b'.



## 5 Семантические действия

1.  $S \rightarrow A a S$
2.  $S \rightarrow b$
3.  $A \rightarrow C A b$
4.  $A \rightarrow B$
5.  $B \rightarrow c S a$
6.  $B \rightarrow \epsilon$
7.  $C \rightarrow a$
8.  $C \rightarrow b$

Семантические действия формируют фрагменты Haskell-кода, которые:

- Генерируют точки на сетке 400x400 пикселей с разными цветами:
  - Синие точки для терминала 'b' (вероятность 0.3)
  - Зеленые точки для терминала 'c' (вероятность 0.7)
  - Оранжевые точки для терминалов 'a' и 'b' в правиле C (вероятность 0.5)
- Создают случайные стены (40 штук) на сетке, избегая пересечения с точками
- Находят кратчайший путь между соседними точками с использованием:
  - Окрестности Мура (8 соседних клеток)
  - Алгоритма поиска в ширину (BFS)
  - Визуализации пути с анимацией

**Пример цепочки языка:** `bbcabbab $`

$$\begin{aligned} S' &\Rightarrow S \$ \\ &\Rightarrow A a S \$ \\ &\Rightarrow C A b a S \$ \quad (\text{правило } A \rightarrow C A b) \\ &\Rightarrow \underbrace{b}_{C \rightarrow b} A b a S \$ \\ &\Rightarrow b b a S \$ \quad (\text{правило } A \rightarrow \epsilon) \\ &\Rightarrow b b a \underbrace{b}_{S \rightarrow b} \$ \end{aligned}$$

При таком выводе анализатор выполняет семантические действия:

- Генерирует оранжевую точку (правило  $C \rightarrow b$ )
- Генерирует синюю точку (правило  $S \rightarrow b$ )
- Создает случайные стены на сетке
- Находит кратчайший путь между точками
- Визуализирует результат на экране

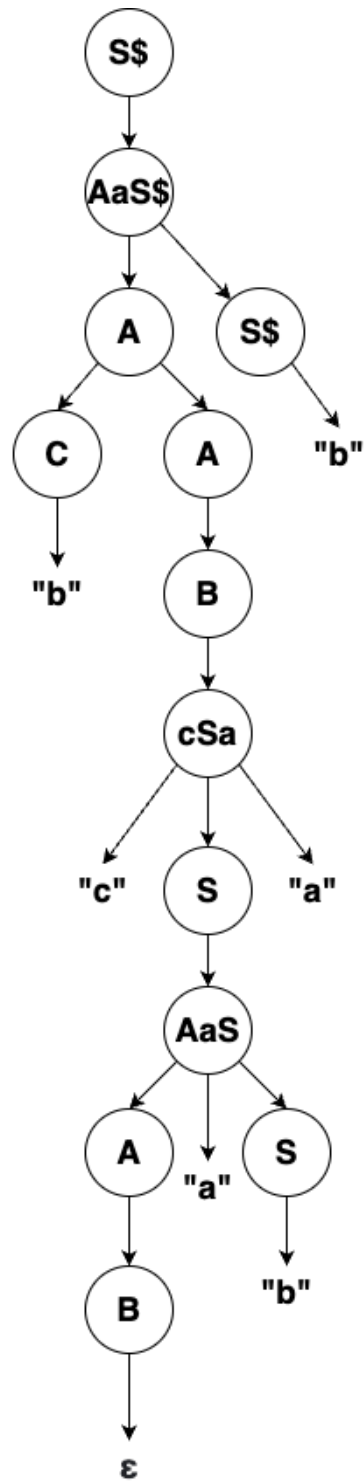


Рис. 3: Дерево для цепочки bbcbabbab \$

Если рассматривать стек, и цепочку - bbcbabbab:

Шаг	Стек (сверху → вниз)	Оставшийся ввод
0	S' \$	bbcabbab \$
1	S \$	bbcabbab \$
2	A a S \$	bbcabbab \$
3	C A b a S \$	bbcabbab \$
4	b A b a S \$	b bcbabbab \$
5	b b a S \$	b bcbabbab \$
6	b b a b \$	b bcbabbab \$
7	b b a \$	c babbab \$
8	b b a B \$	c babbab \$
9	b b a c S a \$	c babbab \$
10	b b a c b a \$	b abbab \$
11	b b a c b \$	a bbab \$
12	b b a c \$	b bab \$
13	b b a \$	b ab \$
14	b b \$	a b \$
15	b \$	b \$
16	\$	\$
17	стек пуст	достиг конца

В столбце **Стек** верхняя строка — вершина. Красным выделен текущий символ входа, который либо сравнивается с вершиной, либо определяет выбор продукции.

Разбор завершается на шаге 17: стек опустел, входная строка полностью обработана — цепочка **bbcabbab \$** принята.

## 6 Программная реализация

Для реализации LL(1)-анализатора был создан cabal проект со следующей структурой:

- Lib.hs: модуль с реализацией LL(1)-анализатора и генерации точек;
- Main.hs: GUI-интерфейс программы;
- Язык программирования: Haskell
- Конфигурация языка: Haskell2010
- Среда разработки: Cursor IDE

Файл Lib.hs:

```
1 {-# LANGUAGE LambdaCase #-}
2 {-# LANGUAGE ParallelListComp #-}
3 {-# OPTIONS_GHC -Wall #-}
4
5 module Lib
6   ( NonTerm(..)
7   , Production(..)
8   , grammar
9   , firstSets, followSets, parseTable
10  , printFirstSets, printFollowSets, printParseTable
11  , parse
12  , genStrings
13  , saveGeneratedStringsToFile
14  , loadGeneratedStrings
15  , genInteract
16  , setup
17  , pause
18  ) where
19
20 import           Data.List           (intercalate, tails, isPrefixOf,
21   minimumBy)
22 import           Data.Map.Strict    (Map)
23 import qualified Data.Map.Strict    as M
24 import           Data.Set           (Set)
25 import qualified Data.Set           as S
26 import           System.IO          (writeFile, readFile, hFlush, stdout)
27 import           System.IO.Unsafe   (unsafePerformIO)
28 import           System.Directory   (doesFileExist)
29 import qualified Graphics.UI.Threepenny as UI
30 import           Graphics.UI.Threepenny.Core
31 import qualified Graphics.UI.Threepenny.Canvas as UI
32 import           Graphics.UI.Threepenny.Canvas (Point(..))
33 import           Control.Monad      (replicateM, forM_, forM, void)
34 import           System.Random      (randomRIO)
35 import           Data.IORef         (newIORef, readIORef, writeIORef)
36 import           Data.Ord            (comparing)
37 import           Data.Maybe         (fromMaybe)
38 import           Control.Concurrent (threadDelay)
39
40 prompt :: String -> IO ()
41 prompt s = putStr s >> hFlush stdout
42
43 pause :: IO ()
44 pause = putStrLn "Press Enter to continue..." >> void getLine
45
46
```

```

47 data NonTerm = S' | S | A | B | C deriving (Eq, Ord, Show)
48 data Symbol  = NT NonTerm | T String | Eps deriving (Eq, Ord, Show)
49
50 data Production = P { lhs :: NonTerm, rhs :: [Symbol], act :: [String] ->
    String }
51
52
53 codeB, codeC, codeAB :: String
54 codeB = " generatePixel 0.3 >= \\p -> print p\n"
55 codeC = " generatePixel 0.7 >= \\p -> print p\n"
56 codeAB = " generatePixel 0.5 >= \\p -> print p\n"
57
58 grammar :: [Production]
59 grammar =
60   [ P S' [NT S, T "$"]           (\\[s,_] -> "main :: IO ()\nmain =
        do\n" <> s)
61   , P S [NT A, T "a", NT S]      (\\[a,_,s] -> a <> s)
62   , P S [T "b"]                  (const codeB)
63   , P A [NT C, NT A, T "b"]      (\\[c,a,_] -> c <> a)
64   , P A [NT B]                   (\\[b] -> b)
65   , P B [T "c", NT S, T "a"]     (\\[_ ,s,_] -> codeC <> s)
66   , P B [Eps]                    (const "")
67   , P C [T "a"]                  (const codeAB)
68   , P C [T "b"]                  (const codeAB)
69   ]
70
71 nonterms :: [NonTerm]
72 nonterms = [S',S,A,B,C]
73
74 type FirstSets = Map NonTerm (Set String)
75 type FollowSets = Map NonTerm (Set String)
76 type Key = (NonTerm,String)
77 type ParseTable = Map Key Production
78
79 aempty :: Set String; aempty = S.empty
80
81 firstSets :: FirstSets
82 firstSets = fixedPoint step (M.fromList [(n,aempty) | n<-nonterms])
83   where
84     step tbl = foldl upd tbl grammar
85     where upd acc (P nt B _) = M.insertWith S.union nt (firstSeq B tbl)
        acc
86
87 firstSeq :: [Symbol] -> FirstSets -> Set String
88 firstSeq [] _ = S.singleton ""
89 firstSeq (Eps:_) _ = S.singleton ""
90 firstSeq (T t:_) _ = S.singleton t
91 firstSeq (NT n:B) tbl =
92   let f = M.findWithDefault aempty n tbl
93   in if S.member "" f then S.delete "" f `S.union` firstSeq B tbl else f
94
95 followSets :: FollowSets
96 followSets = fixedPoint step start
97   where
98     start = M.insert S' (S.singleton "$") (M.fromList [(n,aempty) | n<-
        nonterms])
99     step tbl = foldl proc tbl grammar
100     where
101       proc acc (P nt B _) = foldl (prop nt) acc (zip B (map tail $ tails
        B))
102       prop pnt acc (NT a, suf) =

```

```

103         let fB = firstSeq suf firstSets
104         acc1 = M.insertWith S.union a (S.delete "" fB) acc
105         in if S.member "" fB || null suf
106           then M.insertWith S.union a (M.findWithDefault aempty pnt
107             tbl) acc1
108           else acc1
109     prop _ acc _ = acc
110
111 parseTable :: ParseTable
112 parseTable = foldl add M.empty grammar
113   where
114     add acc p@(P nt B _) = foldl ins acc sels
115       where
116         fB = firstSeq B firstSets
117         sels = if S.member "" fB
118           then S.delete "" fB `S.union` M.findWithDefault aempty nt
119             followSets
120           else fB
121         ins m tok = case M.lookup (nt,tok) m of
122           Nothing -> M.insert (nt,tok) p m
123           Just _   -> error $ "LL(1) conflict at " ++ show (nt,
124             tok)
125
126 type Result = (String,[String])
127
128 splitTokens :: String -> Either String [String]
129 splitTokens s =
130   let invalid = filter (`notElem` "abc") s
131   in if null invalid
132     then Right $ map (:[]) s
133     else Left $ "Invalid characters found: " ++ show invalid
134
135 parseS :: [String] -> [Result]
136 parseS toks = s_b toks ++ s_AaS toks
137   where
138     s_b ("b":xs) = [(codeB,xs)]
139     s_b _        = []
140
141     s_AaS ts =
142       [ (aRes <> sRes, rest2)
143       | (aRes, rest1) <- parseA ts
144       , ("a":afterA) <- [rest1]
145       , (sRes, rest2) <- parseS afterA
146       ]
147
148 parseA :: [String] -> [Result]
149 parseA ts = a_CAb ts ++ a_B ts
150   where
151     a_CAb xs =
152       [ (cRes <> aRes, rest3)
153       | (cRes, rest1) <- parseC xs
154       , (aRes, rest2) <- parseA rest1
155       , ("b":rest3) <- [rest2] ]
156     a_B = parseB
157
158
159 parseB :: [String] -> [Result]
160 parseB ("c":xs) =

```

```

161     [ (codeC <> sRes, rest2)
162     | (sRes, rest1) <- parseS xs
163     , ("a":rest2)    <- [rest1] ]
164 parseB ts = [("",ts)]
165
166
167 parseC :: [String] -> [Result]
168 parseC ("a":xs) = [(codeAB,xs)]
169 parseC ("b":xs) = [(codeAB,xs)]
170 parseC _       = []
171
172
173 parseTokens :: [String] -> Maybe String
174 parseTokens ts =
175     case [c | (c,[]) <- parseS ts] of
176         (r:_) -> Just r
177         _      -> Nothing
178
179 parse :: String -> Either String String
180 parse inp = do
181     tokens <- splitTokens inp
182     case parseTokens tokens of
183         Just c  -> Right c
184         Nothing -> Left "String does not belong to grammar."
185
186
187 printFirstSets :: IO ()
188 printFirstSets = mapM_ pr (M.toList firstSets)
189     where pr (n,s) = putStrLn $ show n ++ " : " ++ show (S.map f s)
190           f "" = "eps"; f x = x
191
192 printFollowSets :: IO ()
193 printFollowSets = mapM_ (\(n,s)->putStrLn $ show n ++ " : " ++ show s)
194                     (M.toList followSets)
195
196 printParseTable :: IO ()
197 printParseTable = mapM_ pr (M.toList parseTable)
198     where pr ((nt,tok),P _ B _) =
199         putStrLn $ show nt ++ " , " ++ tok ++ " ' => " ++ show B
200
201
202 genStrings :: Int -> [[String]]
203 genStrings depth = S.toList $ explore depth [[NT S]]
204     where
205         isNT (NT _) = True; isNT _ = False
206         isT  (T _)  = True; isT  _ = False
207         toTok (T x) = [x];  toTok _ = []
208
209         explore 0 ss = S.fromList [concatMap toTok s | s<-ss, all isT s]
210         explore k ss = explore (k-1) (ss >=> expand)
211
212         expand s = case break isNT s of
213             (_,[]) -> [s]
214             (pre, NT nt:suf) -> [pre ++ B ++ suf | B<-alts nt]
215
216         alts S' = [[NT S, T "$"]]
217         alts S  = [[NT A, T "a", NT S],[T "b"]]
218         alts A  = [[NT C, NT A, T "b"],[NT B]]
219         alts B  = [[T "c", NT S, T "a"],[]]
220         alts C  = [[T "a"],[T "b"]]
221

```

```

222 fixedPoint :: Eq a => (a->a)->a->a
223 fixedPoint f x = let x' = f x in if x'==x then x else fixedPoint f x'
224
225 saveGeneratedStringsToFile :: Int -> IO ()
226 saveGeneratedStringsToFile d = writeFile "generated_strings.txt"
227     . unlines . map concat $ genStrings d
228
229 loadGeneratedStrings :: IO [[String]]
230 loadGeneratedStrings = do
231     e <- doesFileExist "generated_strings.txt"
232     if e then map (map (:[])) . lines <$> readFile "generated_strings.txt"
233     else pure []
234
235 genInteract :: IO ()
236 genInteract = do
237     prompt "Depth? "
238     dStr <- getLine
239     case reads dStr of
240         [(d,"")] | d>0 -> mapM_ (putStrLn . concat) (genStrings d)
241             >> saveGeneratedStringsToFile d
242         _ -> putStrLn "Not a positive integer."
243
244
245 data Pixel = Pixel { x :: Int, y :: Int, color :: String } deriving (Show)
246
247 generatePixel :: Double -> IO Pixel
248 generatePixel prob = do
249     x <- randomRIO (0, 400)
250     y <- randomRIO (0, 400)
251     let r = floor $ prob * 255
252         g = floor $ (1 - prob) * 255
253         b = floor $ (prob + 0.5) * 127
254     pure $ Pixel x y $ "rgb(" ++ show r ++ "," ++ show g ++ "," ++ show b ++
255         ")"
256
257 clearCanvas :: UI.Canvas -> UI ()
258 clearCanvas = UI.clearCanvas
259
260 data PixelPoint = PixelPoint { px :: Int, py :: Int, pointColor :: String
261     }
262     deriving (Eq, Show)
263 type Node = (Int,Int)
264
265 drawPoint :: UI.Canvas -> PixelPoint -> UI ()
266 drawPoint ctx (PixelPoint x y col) = do
267     let gx = (x `div` 10) * 10
268         gy = (y `div` 10) * 10
269     ctx # set' UI.fillStyle (UI.htmlColor col :: UI.FillStyle)
270     _ <- UI.fillRect (fromIntegral gx, fromIntegral gy) 10 10 ctx
271     pure ()
272
273 drawPath :: UI.Canvas -> [Node] -> UI ()
274 drawPath _ [] = pure ()
275 drawPath ctx path = do
276     ctx # set' UI.fillStyle (UI.htmlColor "rgba(0,0,255,0.3)" :: UI.
277         FillStyle)
278     mapM_ (\(x,y)-> UI.fillRect (fromIntegral x,fromIntegral y) 10 10 ctx
279         >> liftIO (threadDelay 100000)) path
280
281 drawGrid :: UI.Canvas -> UI ()

```



```

280 drawGrid ctx = do
281   ctx # set' UI.strokeStyle "lightgray"
282   ctx # set' UI.lineWidth 1
283   forM_ [0,10..400] $ \x -> UI.beginPath ctx >> UI.moveTo (fromIntegral x
284     ,0) ctx
285     >> UI.lineTo (fromIntegral x,400)
286     ctx
287     >> UI.stroke ctx
288   forM_ [0,10..400] $ \y -> UI.beginPath ctx >> UI.moveTo (0,fromIntegral
289     y) ctx
290     >> UI.lineTo (400,fromIntegral y)
291     ctx
292     >> UI.stroke ctx
293
294 data Wall = Wall { wx :: Int, wy :: Int } deriving (Eq, Ord, Show)
295
296 randWall :: Set Node -> IO Wall
297 randWall busy = pick
298   where
299     pick = do
300       gx <- randomRIO (0,39)
301       gy <- randomRIO (0,39)
302       let n = (gx*10,gy*10)
303       if S.member n busy then pick else pure (Wall (fst n) (snd n))
304
305 generateWalls :: Int -> [Node] -> IO [Wall]
306 generateWalls n occ = go n (S.fromList occ) []
307   where
308     go 0 _ acc = pure acc
309     go k busy acc = do
310       w@(Wall x y) <- randWall busy
311       go (k-1) (S.insert (x,y) busy) (w:acc)
312
313 moore :: Node -> [Node]
314 moore (x,y) = [(x+dx,y+dy)
315   | dx<-[-10,0,10], dy<-[-10,0,10], (dx,dy)/=(0,0)]
316
317 inField :: Node -> Bool
318 inField (x,y) = x>=0 && x<=390 && y>=0 && y<=390
319
320 bfs :: Set Node -> Node -> Node -> Maybe [Node]
321 bfs walls start goal = go S.empty (S.singleton start) M.empty
322   where
323     go _ f _ | S.null f = Nothing
324     go vis f prev
325       | goal `S.member` f = Just (restore goal prev)
326       | otherwise = go vis' f' prev'
327     where
328       vis' = vis `S.union` f
329       neigh n = filter (\p->inField p && S.notMember p walls && S.
330         notMember p vis')
331         (moore n)
332       pairs = [ (p,n) | n<-S.toList f, p<-neigh n ]
333       f' = S.fromList (map fst pairs)
334       prev' = foldl (\m (p,n)->M.insertWith (const id) p n m) prev pairs
335       restore n pr | n==start = [n]
336       | otherwise = n : restore (pr M.! n) pr
337
338 data GridCell = GridCell { cellX :: Int, cellY :: Int, cellColor :: String
339   }
340   deriving (Eq, Show)

```

```

335
336 pointsToGrid :: [PixelPoint] -> [GridCell]
337 pointsToGrid = map (\(PixelPoint x y c)
338                   -> GridCell ((x`div`10)*10) ((y`div`10)*10) c)
339
340 single :: String -> IO [PixelPoint]
341 single col = do x<-randomRIO (0,39); y<-randomRIO (0,39)
342               pure [PixelPoint (x*10) (y*10) col]
343
344 double :: String -> IO [PixelPoint]
345 double col = do x1<-randomRIO (0,39); y1<-randomRIO (0,39)
346                x2<-randomRIO (0,39); y2<-randomRIO (0,39)
347                pure [ PixelPoint (x1*10) (y1*10) col
348                      , PixelPoint (x2*10) (y2*10) col ]
349
350 generatePointForToken :: String -> IO [PixelPoint]
351 generatePointForToken "b" = single "blue"
352 generatePointForToken "c" = single "green"
353 generatePointForToken "a" = single "orange"
354 generatePointForToken "ab" = double "red"
355 generatePointForToken _ = pure []
356
357 generatePointsFromGrammar :: String -> IO [PixelPoint]
358 generatePointsFromGrammar = fmap concat
359                             . mapM generatePointForToken
360                             . map (:[])
361
362 generatePattern :: String -> UI.Canvas -> UI ()
363 generatePattern str ctx =
364   case parse str of
365     Left err  -> runFunction $ ffi "alert(%1)" err
366     Right code -> do
367       clearCanvas ctx
368       drawGrid ctx
369       pts <- liftIO $ generatePointsFromGrammar str
370       mapM_ (drawPoint ctx) pts
371
372       let cells = pointsToGrid pts
373           busy   = [(cellX c, cellY c) | c<-cells]
374       walls <- liftIO $ generateWalls 40 busy
375       let wallSet = S.fromList [(wx w, wy w) | w<-walls]
376       forM_ walls $ \(Wall x y) -> do
377         ctx # set' UI.fillStyle (UI.htmlColor "black" :: UI.FillStyle)
378         _ <- UI.fillRect (fromIntegral x, fromIntegral y) 10 10 ctx
379         pure ()
380
381       let gridPts = map (\c -> (cellX c, cellY c)) cells
382           pairs   = zip gridPts (tail gridPts)
383       paths <- forM pairs $ \(a,b) ->
384         case bfs wallSet a b of
385           Nothing -> runFunction (ffi "alert(%1)"
386                                   ("No path between "++show a++
387                                   " and "++show b)) >> pure []
388           Just p   -> pure p
389       mapM_ (drawPath ctx) paths
390       runFunction $ ffi "alert(%1)" code
391
392
393
394
395 setup :: Window -> UI ()
396 setup window = do
397   return window # set title "Pixel Pattern Generator"

```

```

396 canvas <- UI.canvas # set UI.width 400
397                        # set UI.height 400
398                        # set style [("border","1px solid black")]
399
400 input <- UI.input      # set UI.type_ "text"
401                        # set (attr "placeholder") "Enter pattern (e.g.
                        # bcbabbab)"
402 btnGen  <- UI.button  #+ [string "Generate Pattern"]
403 btnClear <- UI.button  #+ [string "Clear"]
404 btnFst  <- UI.button  #+ [string "Show FIRST sets"]
405 btnFol  <- UI.button  #+ [string "Show FOLLOW sets"]
406 btnTbl  <- UI.button  #+ [string "Show parse table"]
407 btnStrs <- UI.button  #+ [string "Generate Strings"]
408 depthInp <- UI.input  # set UI.type_ "number"
409                        # set (attr "placeholder") "Depth (1-50)"
410                        # set (attr "min") "1" # set (attr "max") "50"
411                        # set (attr "value") "3"
412
413 getBody window #+
414   [ column [ element input, element btnGen, element btnClear
415             , element canvas
416             , element btnFst, element btnFol, element btnTbl
417             , element depthInp, element btnStrs ] ]
418
419 on UI.click btnGen    $ const $ do p <- get value input
420                               generatePattern p canvas
421 on UI.click btnClear  $ const $ clearCanvas canvas
422 on UI.click btnFst    $ const $ liftIO (printFirstSets >> pause)
423 on UI.click btnFol    $ const $ liftIO (printFollowSets >> pause)
424 on UI.click btnTbl    $ const $ liftIO (printParseTable >> pause)
425 on UI.click btnStrs   $ const $ do
426   dStr <- get value depthInp
427   case reads dStr of
428     [(d,"")] | d>0 && d<=50 -> liftIO $ do
429       putStrLn ("Depth "++show d++":")
430       mapM_ (putStrLn.concat) (genStrings d)
431       saveGeneratedStringsToFile d
432       pause
433   _ -> runFunction $ ffi "alert(%1)" "Enter depth 1-50!"

```

## Основные функции в Lib.hs:

- **parse :: [String] -> Either String String**

- Принимает список строк (токенов), возвращает Either с ошибкой или сгенерированным кодом
- Основная функция парсинга, которая проверяет принадлежность входной цепочки грамматике
- Реализует LL(1)-анализ входной цепочки и генерирует Haskell-код при успешном разборе

- **firstSets :: FirstSets**

- Map NonTerm (Set String) - отображение нетерминалов в множества их FIRST
- Вычисляет множества FIRST для всех нетерминалов грамматики
- Необходимо для построения таблицы разбора и проверки LL(1)-свойства

- **followSets :: FollowSets**

- Map NonTerm (Set String) - отображение нетерминалов в множества их FOLLOW
- Вычисляет множества FOLLOW для всех нетерминалов грамматики
- Используется при построении таблицы разбора для правил с пустыми выводами
- **parseTable :: ParseTable**
  - Map (NonTerm,String) Production - таблица разбора
  - Строит таблицу разбора на основе множеств FIRST и FOLLOW
  - Определяет, какое правило применять при данном нетерминале и текущем токене
- **genStrings :: Int -> [[String]]**
  - Принимает глубину, возвращает список списков строк
  - Генерирует все возможные цепочки языка до заданной глубины вывода
  - Полезно для тестирования и демонстрации работы грамматики
- **printFirstSets, printFollowSets, printParseTable :: IO ()**
  - Функции без параметров, возвращающие IO ()
  - Выводят на экран множества FIRST, FOLLOW и таблицу разбора соответственно
  - Для отладки и демонстрации работы анализатора

Файл Main.hs:

```

1  {-# LANGUAGE LambdaCase #-}
2  {-# OPTIONS_GHC -Wall #-}
3
4  module Main where
5
6  import           Lib                      (setup)
7  import           Graphics.UI.Threepenny  (defaultConfig, startGUI)
8
9  main :: IO ()
10 main = do
11     putStrLn "Starting Pixel Pattern Generator on http://localhost:8081"
12     startGUI defaultConfig setup

```

- Графический интерфейс (Threepenny):
  - Canvas для отображения сетки и точек
  - Поле ввода для последовательности токенов
  - Кнопки для генерации паттернов и просмотра множеств
- Основные функции:
  - **menu** - создание элементов интерфейса
  - **genInteract** - обработка генерации паттернов
  - **viewSets** - отображение FIRST/FOLLOW множеств

Файл Main.hs реализует графический интерфейс с использованием библиотеки Threepenny:

- Канвас для отображения сетки и точек
- Поле ввода для цепочки
- Кнопки для управления:

- Генерация паттерна
- Очистка канваса
- Просмотр множеств FIRST/FOLLOW
- Просмотр таблицы разбора
- Генерация тестовых цепочек

## 7 Результаты программы

Ниже на рисунках 4 - 10 представлены результаты работы программы.

```
S' : fromList ["a","ab","b","c"]
S  : fromList ["a","ab","b","c"]
A  : fromList ["ab","c","epsilon"]
B  : fromList ["c","epsilon"]
C  : fromList ["ab"]
```

Рис. 4: Множества FIRST для нетерминалов грамматики

```
S' : fromList ["$"]
S  : fromList ["$","a"]
A  : fromList ["a","b"]
B  : fromList ["a","b"]
C  : fromList ["ab","b","c"]
```

Рис. 5: Множества FOLLOW для нетерминалов грамматики

На рисунке 6 показана таблица разбора LL(1)-анализатора.

```
Depth? 15
a a a a b
a a a ab b a b
a a a b
a a a c b a a b
a a ab ab b b a b
a a ab ab c b a b b a b
a a ab b a a b
a a ab b a b
a a ab c b a b a b
a a b
a a c a b a a b
a a c b a a a b
a a c b a a b
a a c b a a c b a a b
a a c c b a a b a a b
a ab ab ab ab b b b b a b
a ab ab ab b b b a b
```

Рис. 7: Генерация цепочек

```

S', 'a' => [NT S,T "$"]
S', 'ab' => [NT S,T "$"]
S', 'b' => [NT S,T "$"]
S', 'c' => [NT S,T "$"]
S, 'a' => [NT A,T "a",NT S]
S, 'ab' => [NT A,T "a",NT S]
S, 'b' => [T "b"]
S, 'c' => [NT A,T "a",NT S]
A, 'a' => [NT B]
A, 'ab' => [NT C,NT A,T "b"]
A, 'b' => [NT B]
A, 'c' => [NT B]
B, 'a' => [Eps]
B, 'b' => [Eps]
B, 'c' => [T "c",NT S,T "a"]
C, 'ab' => [T "ab"]

```

Рис. 6: Таблица разбора LL(1)-анализатора

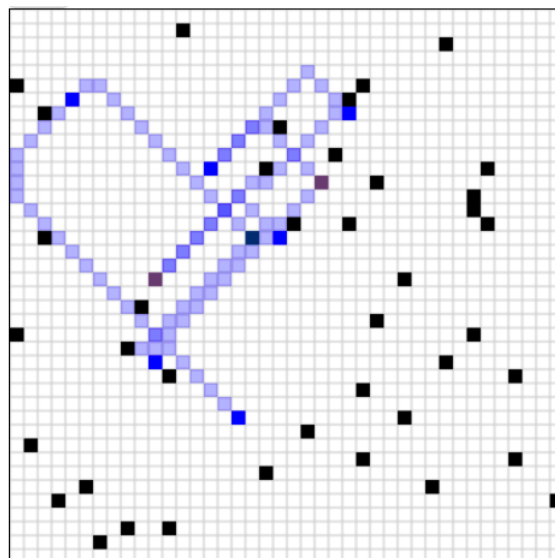


Рис. 8: Начало работы программы с цепочкой bbcabbab



Рис. 9: Результат цепочки bbcbabbab в текстовом виде

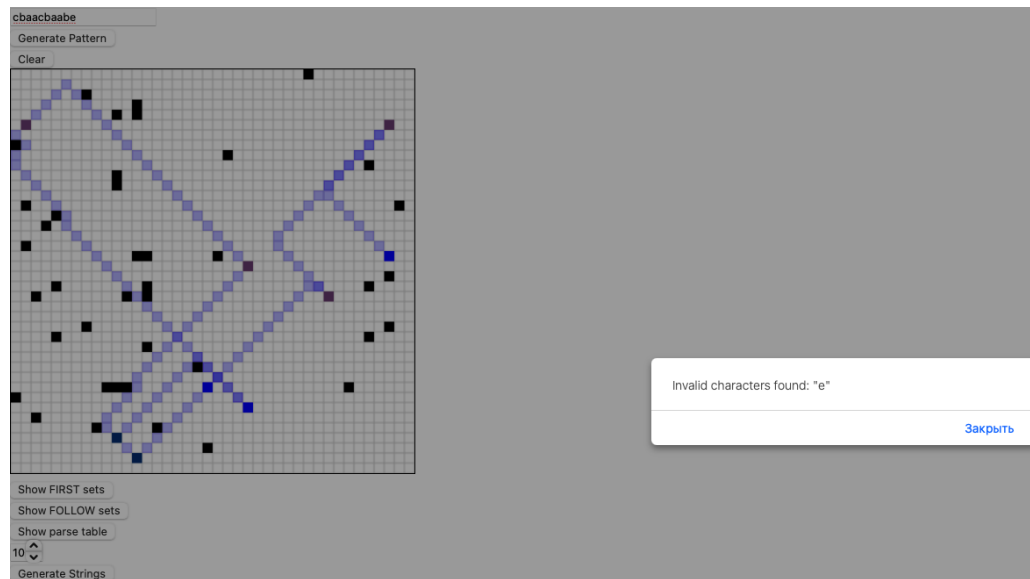


Рис. 10: Ошибка при вводе цепочки (сбаасбааб - верно, е - неверно)



## Заключение

В данной лабораторной работе была рассмотрена контекстно-свободная грамматика (КС-грамматика) и реализован полный цикл построения LL(1)-анализатора для неё. Были вычислены множества **FIRST** и **FOLLOW**. На их основе сформирована таблица выбора (look-up table). Отсутствие пересечений между множествами выбора альтернатив одного нетерминала подтвердило, что грамматика принадлежит классу LL(1).

Каждой продукции назначено семантическое действие, интерпретируемое как генерация точек на сетке и поиск пути между ними: `generatePixel 0.3` для синих точек, `generatePixel 0.7` для зеленых и `generatePixel 0.5` для оранжевых точек, с последующим поиском кратчайшего пути между ними с использованием окрестности Мура. Для разбора был реализован детерминированный LL(1)-предиктивный анализатор.

В ходе работы был реализован LL(1) парсер для заданной грамматики.

- **Плюсы:**

- Сохранена грамматика LL(1).
- Разбирает входные цепочки и генерирует код для создания точек.
- Визуализирует точки на сетке и находит кратчайший путь между ними.

- **Минусы**

- Лексер тривиален: терминал **ab** должен вводиться как единая лексема, иначе разбор завершится ошибкой; для «настоящего» языка это потребует более гибкого токенизатора.
- Пришлось редактировать грамматику для корректного разбора. Не был реализован парсер для LL(3).

- **Дополнения**

1. Усовершенствовать поиск минимальных путей алгоритмом A\*.
2. Добавить возможность вводить иные грамматики.
3. Добавить возможность добавления "черных" точек в соответствии с грамматикой.

## Список литературы

1. Ю.Г. Карпов, Теория и технология программирования. Основы построения трансляторов. СПб.: БХВ-Петербург, 2005.
2. А.В. Востров, "Математическая логика и теория автоматов". Распознавание КС-языков и трансляция. [Электронный ресурс], URL: <https://tema.spbstu.ru/userfiles/files/courses/2018-compilers> (дата обращения: 14.05.2025)