

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 №3
«КС грамматика подмножества естественного языка»
Вариант **Pluperfect**

Студент: _____

Салимли Айзек Мухтар Оглы

Преподаватель: _____

Востров Алексей Владимирович

«____» _____ 20__ г.

Санкт-Петербург, 2025

Содержание

Введение	3
1 Описание грамматик и их математическая модель	4
1.1 Plusquamperfekt (PQ)	4
1.2 Präsens (PR)	4
1.3 Modal + Infinitiv (MI)	5
1.4 Azeri Present (AZ)	5
1.5 Правило пунктуации	6
1.6 LL(1)-свойства (пример для PQ)	6
1.7 Переход к КС-грамматикам	7
2 Формальное определение CFG	8
3 Дерево разбора для Pluperfect	9
4 Особенности реализации	10
5 ϵ-переход в программной реализации	11
6 Результаты работы	17
Заключение	20
Приложение А. Порождающая грамматика (Pluperfect)	21

Введение

Цель данной работы — составить грамматику для подмножества предложений Немецкого языка, времени: Pluperfect. На основе созданной грамматики требуется реализовать алгоритм распознавания введенного предложения и генерацию случайного предложения, принадлежащего грамматике.

Необходимо:

1. описать контекстно-свободную грамматику (КСГ) времени Plusquamperfekt;
2. реализовать:
 - генератор случайных предложений Pluperfect;
 - распознаватель предложений Pluperfect;
 - дополнительно:
 - Prasens
 - Modal + Infinitiv
 - Азербайджанское предложение «ADV + OBJ» Present.

1 Описание грамматик и их математическая модель

Сокращения. **PQ** — Plusquamperfekt, **PR** — Präsens, **MI** — «Modal + Infinitiv», **AZ** — Azeri Present.

Во всех BNF-фрагментах ниже

- вертикальная черта «|» — альтернатива;
- ϵ — пустая цепочка;
- после структурных правил явно даны *лексические* productions с 7–8 примерами терминалов каждой части речи.

1.1 Plusquamperfekt (PQ)

$SUBJ\ AUX_{Prät.}\ [ADV]\ [DET\ N]\ PPART$

$\langle S \rangle ::= \langle Pronoun \rangle \langle Aux \rangle \langle Mid \rangle \langle PPart \rangle$
 $\langle Mid \rangle ::= \epsilon \mid \langle Adverb \rangle \mid \langle ObjGrp \rangle \mid \langle Adverb \rangle \langle Comma \rangle \langle ObjGrp \rangle$
 $\langle ObjGrp \rangle ::= \langle Det \rangle \langle Noun \rangle$
 $\langle Pronoun \rangle ::= \text{ich} \mid \text{du} \mid \text{er} \mid \text{sie} \mid \text{es} \mid \text{wir} \mid \text{ihr} \mid \text{Sie}$
 $\langle Aux \rangle ::= \text{hatte} \mid \text{hattest} \mid \text{hatten} \mid \text{hattet} \mid \text{war} \mid \text{warst} \mid \text{waren}$
 $\langle Adverb \rangle ::= \text{heute} \mid \text{gestern} \mid \text{schon} \mid \text{oft} \mid \text{morgen} \mid \text{damals} \mid \text{später}$
 $\langle Det \rangle ::= \text{einen} \mid \text{eine} \mid \text{ein} \mid \text{den} \mid \text{die} \mid \text{das} \mid \text{diesen} \mid \text{jeden}$
 $\langle Noun \rangle ::= \text{Brief} \mid \text{Apfel} \mid \text{Hund} \mid \text{Katze} \mid \text{Buch} \mid \text{Auto} \mid \text{Haus} \mid \text{Zeitung}$
 $\langle PPart \rangle ::= \text{gesehen} \mid \text{gegessen} \mid \text{gemacht} \mid \text{gegangen} \mid \text{geschlafen} \mid \text{geschrieben} \mid \text{gesagt} \mid \text{gefunden}$
 $\langle Comma \rangle ::= \text{“,”}$

Во всех правилах слева один нетерминал $\Rightarrow G_{PQ} \in \text{CFG}$.

1.2 Präsens (PR)

$SUBJ\ VERB\ [ADV]\ OBJ$

$\langle S \rangle ::= \langle Pronoun \rangle \langle Verb \rangle \langle Tail \rangle$
 $\langle Tail \rangle ::= \langle Obj \rangle \mid \langle Adverb \rangle \langle Comma \rangle \langle Obj \rangle$
 $\langle Pronoun \rangle ::= \text{ich} \mid \text{du} \mid \text{er} \mid \text{sie} \mid \text{es} \mid \text{wir} \mid \text{ihr} \mid \text{Sie}$
 $\langle Verb \rangle ::= \text{sehe} \mid \text{siehst} \mid \text{sieht} \mid \text{essen} \mid \text{isst} \mid \text{macht} \mid \text{geht} \mid \text{schläft}$
 $\langle Adverb \rangle ::= \text{heute} \mid \text{gestern} \mid \text{schon} \mid \text{oft} \mid \text{morgen} \mid \text{selten} \mid \text{immer}$
 $\langle Obj \rangle ::= \text{den_Apfel} \mid \text{das_Buch} \mid \text{die_Katze} \mid \text{den_Hund} \mid \text{mich} \mid \text{dich} \mid \text{ihn} \mid \text{nichts}$
 $\langle Comma \rangle ::= \text{“,”}$

Каждое правило имеет форму $A \rightarrow \alpha \Rightarrow G_{PR} - \text{KC}$.

1.3 Modal + Infinitiv (MI)

SUBJ MODAL ADV [OBJ] INF

$\langle S \rangle ::= \langle Pronoun \rangle \langle Modal \rangle \langle Adverb \rangle \langle Rest \rangle$
 $\langle Rest \rangle ::= \langle Comma \rangle \langle Inf \rangle \mid \langle Obj \rangle \langle Comma \rangle \langle Inf \rangle$
 $\langle Pronoun \rangle ::= \text{ich} \mid \text{du} \mid \text{er} \mid \text{sie} \mid \text{es} \mid \text{wir} \mid \text{ihr} \mid \text{Sie}$
 $\langle Modal \rangle ::= \text{will} \mid \text{willst} \mid \text{wollen} \mid \text{kann} \mid \text{kannst} \mid \text{müssen}$
 $\langle Adverb \rangle ::= \text{heute} \mid \text{gestern} \mid \text{schon} \mid \text{oft} \mid \text{morgen} \mid \text{bald} \mid \text{gleich}$
 $\langle Obj \rangle ::= \text{den_Apfel} \mid \text{das_Buch} \mid \text{die_Katze} \mid \text{den_Hund} \mid \text{mich} \mid \text{dich} \mid \text{ihn} \mid \text{nichts}$
 $\langle Inf \rangle ::= \text{sehen} \mid \text{essen} \mid \text{machen} \mid \text{gehen} \mid \text{schlafen} \mid \text{schreiben} \mid \text{sagen} \mid \text{finden}$
 $\langle Comma \rangle ::= \text{","}$

Оptionальный объект вынесен в альтернативу с ϵ , а слева всегда один нетерминал $\Rightarrow G_{MI} \in \text{CFG}$.

1.4 Azeri Present (AZ)

SUBJ ADV OBJ "VERB

$\langle S \rangle ::= \langle AzPron \rangle \langle AzAdv \rangle \langle AzObj \rangle \langle Comma \rangle \langle AzVerb \rangle$
 $\langle AzPron \rangle ::= \text{men} \mid \text{sen} \mid \text{o} \mid \text{biz} \mid \text{siz} \mid \text{onlar}$
 $\langle AzAdv \rangle ::= \text{bugun} \mid \text{dunen} \mid \text{tez-tez} \mid \text{hemishe} \mid \text{indi} \mid \text{sabah} \mid \text{axsham}$
 $\langle AzObj \rangle ::= \text{kitabı} \mid \text{evi} \mid \text{alma} \mid \text{mashini} \mid \text{mektubu} \mid \text{cantani} \mid \text{suyu}$
 $\langle AzVerb \rangle ::= \text{oxuyuram} \mid \text{oxuyursan} \mid \text{oxuyur} \mid \text{yaziram} \mid \text{gedirem} \mid \text{gorurem} \mid \text{aliram} \mid \text{sevirem}$
 $\langle Comma \rangle ::= \text{","}$

Все правила $A \rightarrow \alpha \Rightarrow G_{AZ} \in \text{CFG}$.

1.5 Правило пунктуации

В каждой немецкой грамматике запятая — *отдельный* терминал, задаваемый единственным правилом

$$\langle Comma \rangle \rightarrow “,”$$

и включаемый в структуры где требуется:

$$\mathbf{PQ:} \quad \langle Mid \rangle \rightarrow \langle Adverb \rangle \langle Comma \rangle \langle ObjGrp \rangle,$$

$$\mathbf{MI:} \quad \langle Rest \rangle \rightarrow \langle Comma \rangle \langle Inf \rangle \mid \langle Obj \rangle \langle Comma \rangle \langle Inf \rangle.$$

Тем самым запятая входит в терминальный алфавит T_{PQ}, T_{MI} (T_{PR}, T_{AZ} не содержат её) и контролируется грамматикой, а не «регулярным» квантификатором $[,]?$.

1.6 LL(1)-свойства (пример для PQ)

Таблица 1: FIRST/FOLLOW (фрагмент) для G_{PQ}

Нетерминал	FIRST	FOLLOW
$\langle S \rangle$	Pronoun	«.»
$\langle Mid \rangle$	$\{\epsilon\}$, Adverb, Det	PPart
$\langle ObjGrp \rangle$	Det	PPart, «,»

Для всех альтернатив одного нетерминала $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$ и, если $\beta \Rightarrow^* \epsilon$, $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \emptyset$. Поэтому все четыре грамматики LL(1).

1.7 Переход к КС-грамматикам

X Y $[Z]$ регулярны. Чтобы получить КС-описание \Rightarrow :

1. блоки вынесены в нетерминалы с альтернативой ϵ ;
2. запятая «,» включена в правила как терминал;
3. слева каждого правила ровно один нетерминал.

\Rightarrow Языки являются контекстно-свободными.

2 Формальное определение CFG

Определение. КС-грамматика — четвёрка $G = \langle T, N, S, R \rangle$, где

- T — конечный алфавит терминалов;
- N — конечный алфавит нетерминалов, $T \cap N = \emptyset$;
- $S \in N$ — стартовый символ;
- $R \subseteq N \times (T \cup N)^*$ — конечное множество правил вида $A \rightarrow \alpha$ ($A \in N$).

Для $\phi, \psi \in (T \cup N)^*$ $\phi \Rightarrow \psi$, если $\exists A \rightarrow \alpha \in R : \phi = \pi A \mu, \psi = \pi \alpha \mu$; \Rightarrow^* — транзитивное замыкание, $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$.

Все четыре разработанные грамматики удовлетворяют условию $A \rightarrow \alpha$ и содержат ϵ -продукции, следовательно $G_{PQ}, G_{PR}, G_{MI}, G_{AZ} \in \text{CFG}$.

3 Дерево разбора для Pluperfect

На рисунке 19, показано дерево разбора для грамматики pluperfect. Дерево строилось снизу вверх.

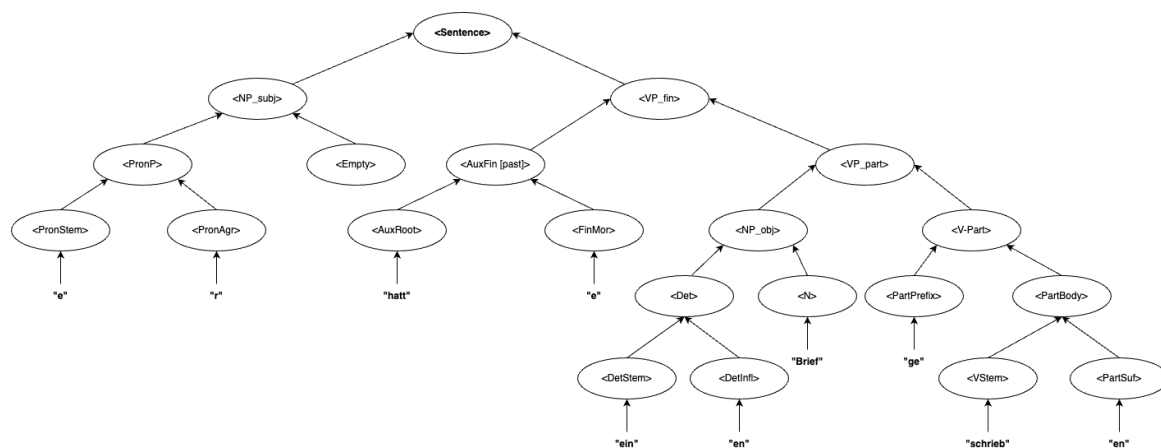


Рис. 1: Дерево разбора: Pluperfect

4 Особенности реализации

Для реализации был создан Cabal-проект, с .cabal 8.0, код написан на языке Haskell, с спецификацией Haskell2010. Проект был создан и реализован в среде разработки VSCode. В проекте файлы:

- Main.hs: Файл меню
- Lib.hs: Логика программы
- **Генератор** — функция `deriveRandomPluperfect` (Haskell).
- **Распознаватель** — функция `isPluperfect` проверяет порядок токенов и принадлежность к лексиконам.
- Для остальных временных форм (PrAAsens, Modal+Infinitiv, Azeri) добавлено строгое лицо-число согласование.

5 ϵ -переход в программной реализации

В Haskell-коде (файл `Lib.hs`) опциональные фрагменты грамматики моделируются *функционально*:

- Для генератора предложений используется `randMaybe a → IO (Maybe a)`. Если функция возвращает `Nothing`, соответствующий блок `($advM, $objM)` ***просто не вставляется*** в список токенов. Это эквивалент выводу по правилу с ϵ справа.
- При проверке предложения валидаторы интерпретируют «пустую» позицию через отдельные ветви `validMid []` / `validTail []`. Такой разбор соответствует переходу $\langle Mid \rangle \Rightarrow \epsilon$, $\langle Tail \rangle \Rightarrow \epsilon$, $\langle Rest \rangle \Rightarrow "$ $\langle Inf \rangle$ в соответствующих BNF.

Листинг 1: Фрагмент реализации ϵ -перехода

```
1 advM <- randMaybe advDE
2 objM <- randMaybe (,) <> detsAcc <*> nounsAcc
3 ...
4 core = case (advM, objM) of                -- epsilon-vetka!!
5   (Nothing, Nothing)    -> [subj, aux, pp]    -- <Mid> -> epsilon!!
6   (Just a, Nothing)     -> [subj, aux, a, pp]
7   ...
8
9 -- CHECK !!! <Mid> ::= epsilon | Adv | ObjGrp | Adv "," ObjGrp
10 validMid []              = True              -- epsilon!!
11 validMid [adv]           = adv `elem` advDE
12 validMid [det, noun]     = ...
```

Таким образом, **каждое** появление `Maybe` в коде соответствует ϵ -производству в математической грамматике, а варианты `Nothing` / пустой список — это явная реализация ϵ -перехода в дереве вывода.

Ниже представлен листинг `Main.hs` и `Lib.hs` соответственно. **Main.hs**

```
1 module Main where
2 import Lib
3 import System.IO (hFlush, stdout)
4
5 main :: IO ()
6 main = do
7   putStr ""
8   menuLoop
9
10 menuLoop :: IO ()
11 menuLoop = do
12   putStr "> "
13   hFlush stdout
14   choice <- getLine
15   case choice of
16     "1" -> deriveAndShow deriveRandomPluperfect >> menuLoop
17     "2" -> askAndCheck "Plusquamperfekt" isPluperfect >> menuLoop
18     "3" -> deriveAndShow deriveRandomPresent >> menuLoop
19     "4" -> askAndCheck "Prasens" isPresent >> menuLoop
20     "5" -> deriveAndShow deriveRandomModal >> menuLoop
21     "6" -> askAndCheck "Modal + Infinitiv" isModal >> menuLoop
22     "7" -> deriveAndShow deriveRandomAzeri >> menuLoop
23     "8" -> askAndCheck "Azerbaijan" isAzeri >> menuLoop
24     "0" -> putStrLn "Sagolun / Aufiderzein!"
25     "" -> putStrLn "Sagolun / Aufiderzein!"
26     _ -> putStrLn "Choose menu nums!!!" >> menuLoop
27   where
```

```

28     deriveAndShow gen = gen >>= putStrLn . ("Generated: " ++)
29     askAndCheck tag predF = do
30         putStrLn $ "Enter sentence (" ++ tag ++ "):"
31         putStr " >> "
32         hFlush stdout
33         sent <- getLine
34         putStrLn $ if predF sent then "Good" else "Not"

```

1. - Генерация Pluperfect
2. - Проверка на Pluperfect
3. - Генерация Präsens
4. - Проверка на Präsens
5. - Генерация Modal + Infinitiv
6. - Проверка на Modal + Infinitiv
7. - Генерация азербайджанский (ADV + OBJ) Present
8. - Проверка на азербайджанский
9. - Выход

Lib.hs:

```

1  module Lib (
2      deriveRandomPluperfect, deriveRandomPresent, deriveRandomModal,
3      deriveRandomAzeri,
4      isPluperfect, isPresent, isModal, isAzeri
5  ) where
6
7  import System.Random (randomRIO)
8  import Data.Maybe    (isJust)
9  import Data.List     (find)
10
11 type Terminal = String
12
13 randFrom :: [a] -> IO a
14 randFrom xs = (xs !!) <$> randomRIO (0, length xs - 1)
15
16 randMaybe :: [a] -> IO (Maybe a)
17 randMaybe xs = do
18     flag <- randomRIO (0 :: Int, 1)
19     if flag == 1 then Just <$> randFrom xs else pure Nothing
20
21 maybeToList :: Maybe a -> [a]
22 maybeToList (Just x) = [x]
23 maybeToList Nothing = []
24
25 pronouns :: [Terminal]
26 pronouns = ["ich", "du", "er", "sie", "es", "wir", "ihr", "Sie"]
27
28 personIndexDE :: Terminal -> Int
29 personIndexDE p
30 | p == "ich" = 0
31 | p == "du" = 1
32 | p `elem` ["er", "sie", "es"] = 2
33 | p == "wir" = 3
34 | p == "ihr" = 4
35 | p == "Sie" = 5

```

```

35 | otherwise = error "unknown pronoun"
36
37 adverbs :: [Terminal]
38 adverbs = ["heute", "gestern", "schon", "oft", "morgen"]
39
40 auxsPlu :: [Terminal]
41 auxsPlu = ["hatte", "hattest", "hatten", "hattet", "war", "warst", "waren", "wart"]
42
43 pparts :: [Terminal]
44 pparts = ["gesehen", "gegessen", "gemacht", "gegangen", "geschlafen", "geschrieen", "gesagt"]
45
46 determinersAcc :: [Terminal]
47 determinersAcc = ["einen", "eine", "ein", "den", "die", "das"]
48
49 nounsAcc :: [Terminal]
50 nounsAcc = ["Brief", "Apfel", "Hund", "Katze", "Buch", "Auto", "Haus", "Zeitung"]
51
52 randMaybeObject :: IO (Maybe (Terminal, Terminal))
53 randMaybeObject = do
54   flag <- randomRIO (0 :: Int, 1)
55   if flag == 1 then do
56     det <- randFrom determinersAcc
57     noun <- randFrom nounsAcc
58     pure $ Just (det, noun)
59   else
60     pure Nothing
61
62 deriveRandomPluperfect :: IO String
63 deriveRandomPluperfect = do
64   subj <- randFrom pronouns
65   aux <- randFrom auxsPlu
66   advM <- randMaybe adverbs
67   objM <- randMaybeObject
68   ppar <- randFrom pparts
69   let objTokens = maybe [] (\(d, n) -> [d, n]) objM
70   pure . unwords $ [subj, aux] ++ maybeToList advM ++ objTokens ++ [ppar]
71
72 isPluperfect :: String -> Bool
73 isPluperfect str =
74   case words str of
75     (subj : aux : rest)
76       | subj `elem` pronouns
77       , aux `elem` auxsPlu
78       , not (null rest)
79       , let ppart = last rest
80       , ppart `elem` pparts
81       , validMid (init rest)
82       -> True
83     _ -> False
74   where
85     validMid [] = True
86     validMid [adv] = adv `elem` adverbs
87     validMid [det, noun] = det `elem` determinersAcc && noun `elem` nounsAcc
88     validMid [adv, det, noun] = adv `elem` adverbs && det `elem` determinersAcc && noun `elem` nounsAcc
89     validMid _ = False
90
91 data GVerb = GVerb { gBase :: Terminal, gForms :: [Terminal] }
92

```

```

93 verbsPresent :: [GVerb]
94 verbsPresent =
95   [ GVerb "sehen"    ["sehe","siehst","sieht","sehen","seht","sehen"]
96   , GVerb "essen"    ["esse","isst","isst","essen","esst","essen"]
97   , GVerb "machen"   ["mache","machst","macht","machen","macht","machen"]
98   , GVerb "gehen"    ["gehe","gehst","geht","gehen","geht","gehen"]
99   , GVerb "schlafen" ["schlafe","schlAAfst","schlAAft","schlafen","schlaft",
    "schlafen"]
100 ]
101
102 objects :: [Terminal]
103 objects = ["den_Apfel","das_Buch","die_Katze","den_Hund","mich","dich","ihn",
    "sie","es","nichts"]
104
105 conjugateDE :: Terminal -> GVerb -> Terminal
106 conjugateDE pron v = gForms v !! personIndexDE pron
107
108 deriveRandomPresent :: IO String
109 deriveRandomPresent = do
110   subj <- randFrom pronouns
111   v      <- randFrom verbsPresent
112   advM <- randMaybe adverbs
113   obj   <- randFrom objects
114   let verb = conjugateDE subj v
115   pure . unwords $ [subj, verb] ++ maybeToList advM ++ [obj]
116
117 isPresent :: String -> Bool
118 isPresent s =
119   case words s of
120     (subj : verb : rest)
121       | subj `elem` pronouns
122       , let idx = personIndexDE subj
123       , isJust (find (\v -> gForms v !! idx == verb) verbsPresent)
124       , validTail rest
125       -> True
126     _ -> False
127   where
128     validTail [obj]      = obj `elem` objects
129     validTail [adv, obj] = adv `elem` adverbs && obj `elem` objects
130     validTail _          = False
131
132 data MVerb = MVerb { mBase :: Terminal, mForms :: [Terminal] }
133
134 modals :: [MVerb]
135 modals =
136   [ MVerb "wollen" ["will","willst","will","wollen","wollt","wollen"]
137   , MVerb "konnen" ["kann","kannst","kann","konnen","konnt","konnen"]
138   , MVerb "mussen" ["muss","musst","muss","mussen","musst","mussen"]
139   ]
140
141 infinitives :: [Terminal]
142 infinitives = ["sehen","essen","machen","gehen","schlafen","schreiben","sagen"]
143
144 conjugateModal :: Terminal -> MVerb -> Terminal
145 conjugateModal p m = mForms m !! personIndexDE p
146
147 deriveRandomModal :: IO String
148 deriveRandomModal = do
149   subj <- randFrom pronouns
150   mverb <- randFrom modals

```

```

151   adv  <- randFrom adverbs
152   objM <- randMaybe objects
153   inf  <- randFrom infinitives
154   let modalForm = conjugateModal subj mverb
155   pure . unwords $ [subj, modalForm, adv] ++ maybeToList objM ++ [inf]
156
157   isModal :: String -> Bool
158   isModal s =
159     case words s of
160       (subj : modal : adv : rest)
161         | subj `elem` pronouns
162         , adv  `elem` adverbs
163         , let idx = personIndexDE subj
164         , isJust (find (\m -> mForms m !! idx == modal) modals)
165         , validTail rest
166         -> True
167       _ -> False
168   where
169     validTail [inf]      = inf `elem` infinitives
170     validTail [obj, inf] = obj `elem` objects && inf `elem` infinitives
171     validTail _         = False
172
173   pronAze :: [Terminal]
174   pronAze = ["men", "sen", "o", "biz", "siz", "onlar"]
175
176   personIndexAZ :: Terminal -> Int
177   personIndexAZ p = case p of
178     "men"    -> 0
179     "sen"    -> 1
180     "o"      -> 2
181     "biz"    -> 3
182     "siz"    -> 4
183     "onlar"  -> 5
184     _        -> error "unknown Azeri pronoun"
185
186   data AzVerb = AzVerb { azRoot :: Terminal, azForms :: [Terminal] }
187
188   azVerbs :: [AzVerb]
189   azVerbs =
190     [ AzVerb "oxu" ["oxuyuram", "oxuyursan", "oxuyur", "oxuyuruq", "oxuyursunuz", "oxuyurlar"]
191     , AzVerb "yaz" ["yaziram", "yazirsan", "yazir", "yaziriq", "yazirsiniz", "yazirlar"]
192     , AzVerb "get" ["gedirem", "gedirsən", "gedir", "gedirik", "gedirsiniz", "gedirler"]
193     , AzVerb "gor" ["gorurem", "gorursən", "gorur", "goruruk", "gorursunuz", "gorurler"]
194     , AzVerb "al"  ["aliram", "alirsan", "alir", "aliriq", "alirsiniz", "alirlar"]
195     , AzVerb "sev" ["sevirem", "sevirsən", "sevir", "sevirik", "sevirsiniz", "sevirler"]
196   ]
197
198   adverbsAze :: [Terminal]
199   adverbsAze = ["bugun", "dunen", "tez-tez", "hemishe"]
200
201   objectsAze :: [Terminal]
202   objectsAze = ["kitabi", "evi", "alma", "mashini", "mektubu"]
203
204   conjugateAZ :: Terminal -> AzVerb -> Terminal
205   conjugateAZ p v = azForms v !! personIndexAZ p
206

```

```

207 deriveRandomAzeri :: IO String
208 deriveRandomAzeri = do
209   subj <- randFrom pronAze
210   adv  <- randFrom adverbsAze
211   obj  <- randFrom objectsAze
212   v    <- randFrom azVerbs
213   let verb = conjugateAZ subj v
214   pure . unwords $ [subj, adv, obj, verb]
215
216 isAzeri :: String -> Bool
217 isAzeri s =
218   case words s of
219     [subj, adv, obj, verb]
220     | subj `elem` pronAze
221     , adv `elem` adverbsAze
222     , obj `elem` objectsAze
223     , let idx = personIndexAZ subj
224     , isJust (find (\v -> azForms v !! idx == verb) azVerbs)
225     -> True
226     _ -> False

```

- `randFrom :: [a] -> IO a` — принимает список, возвращает случайный элемент; универсальный выбор.
- `randMaybe :: [a] -> IO (Maybe a)` — тот же список, выдаёт `Just x` или `Nothing`; вводит случайную опциональность.
- `maybeToList :: Maybe a -> [a]` — преобразует `Nothing/Just x` в `[]/[x]`; упрощает конкатенации токенов.
- `conjugateDE :: Terminal -> GVerb -> Terminal` — местоимение + структура глагола -> согласованная форма `PrAAsens`.
- `conjugateModal :: Terminal -> MVerb -> Terminal` — то же, но для модальных глаголов.
- `conjugateAZ :: Terminal -> AzVerb -> Terminal` — согласует азербайджанский глагол с подлежащим.
- `deriveRandomPluperfect :: IO String` — генерирует случайное предложение `Plusquamperfekt`.
- `isPluperfect :: String -> Bool` — проверяет, принадлежит ли строка грамматике `Pluperfect`.
- `deriveRandomPresent :: IO String` — создаёт предложение в `PrAAsens` с правильным согласованием.
- `isPresent :: String -> Bool` — распознаёт `PrAAsens`-строки.
- `deriveRandomModal :: IO String` — генерирует структуру «Modal + Infinitiv».
- `isModal :: String -> Bool` — валидирует модальные предложения.
- `deriveRandomAzeri :: IO String` — выдаёт азербайджанское предложение вида *SUBJ ADV OBJ VERB*.
- `isAzeri :: String -> Bool` — проверка соответствия азербайджанской грамматике.

6 Результаты работы

Ниже представлены результаты работы программы на рисунках 1-10. Меню, генерация и проверки соответственно.

Pluperfect:

```
Меню:
 1 - Сгенерировать (Plusquamperfekt) [DE]
 2 - Проверить своё (Plusquamperfekt) [DE]
 3 - Сгенерировать (Präsens) [DE]
 4 - Проверить своё (Präsens) [DE]
 5 - Сгенерировать (Modal + Infinitiv) [DE]
 6 - Проверить своё (Modal + Infinitiv) [DE]
 7 - Сгенерировать (Азербайджанский: ADV+OBJ Present) [AZ]
 8 - Проверить своё (Азербайджанский: ADV+OBJ Present) [AZ]
 0 - Выход
> 1
Сгенерировано: Du hattest den Haus gemacht?
```

Рис. 2: Pluperfect.

Проверка Pluperfect:

```
Меню:
 1 - Сгенерировать (Plusquamperfekt) [DE]
 2 - Проверить своё (Plusquamperfekt) [DE]
 3 - Сгенерировать (Präsens) [DE]
 4 - Проверить своё (Präsens) [DE]
 5 - Сгенерировать (Modal + Infinitiv) [DE]
 6 - Проверить своё (Modal + Infinitiv) [DE]
 7 - Сгенерировать (Азербайджанский: ADV+OBJ Present) [AZ]
 8 - Проверить своё (Азербайджанский: ADV+OBJ Present) [AZ]
 0 - Выход
> 2
Введите предложение (Plusquamperfekt):
>> Du hattest den Haus gemacht?
✓ Всё верно!
```

Рис. 3: Проверка Pluperfect.

Ошибка - не в алфавита:

```
Меню:
 1 - Сгенерировать (Plusquamperfekt) [DE]
 2 - Проверить своё (Plusquamperfekt) [DE]
 3 - Сгенерировать (Präsens) [DE]
 4 - Проверить своё (Präsens) [DE]
 5 - Сгенерировать (Modal + Infinitiv) [DE]
 6 - Проверить своё (Modal + Infinitiv) [DE]
 7 - Сгенерировать (Азербайджанский: ADV+OBJ Present) [AZ]
 8 - Проверить своё (Азербайджанский: ADV+OBJ Present) [AZ]
 0 - Выход
> 2
Введите предложение (Plusquamperfekt):
>> шказкзь.
✗ Есть слово вне алфавита: шказкзь
```

Рис. 4: Есть слова вне алфавита.

Ошибка - не в грамматике:

```
Меню:
 1 - Сгенерировать (Plusquamperfekt) [DE]
 2 - Проверить своё (Plusquamperfekt) [DE]
 3 - Сгенерировать (Präsens) [DE]
 4 - Проверить своё (Präsens) [DE]
 5 - Сгенерировать (Modal + Infinitiv) [DE]
 6 - Проверить своё (Modal + Infinitiv) [DE]
 7 - Сгенерировать (Азербайджанский: ADV+OBJ Present) [AZ]
 8 - Проверить своё (Азербайджанский: ADV+OBJ Present) [AZ]
 0 - Выход
> 2
Введите предложение (Plusquamperfekt):
>> Du den Haus gemacht?
❌ Не принадлежит грамматике
```

Рис. 5: Не пренадлежит грамматике.

Генерация предложения на азербайджанском Present:

```
Меню:
 1 - Сгенерировать (Plusquamperfekt) [DE]
 2 - Проверить своё (Plusquamperfekt) [DE]
 3 - Сгенерировать (Präsens) [DE]
 4 - Проверить своё (Präsens) [DE]
 5 - Сгенерировать (Modal + Infinitiv) [DE]
 6 - Проверить своё (Modal + Infinitiv) [DE]
 7 - Сгенерировать (Азербайджанский: ADV+OBJ Present) [AZ]
 8 - Проверить своё (Азербайджанский: ADV+OBJ Present) [AZ]
 0 - Выход
> 7
Сгенерировано: 0 dunen mashini, görür?
```

Рис. 6: Azerbaijan Present.

Неверное предложение Modal+Infinitiv:

```
Меню:
 1 - Сгенерировать (Plusquamperfekt) [DE]
 2 - Проверить своё (Plusquamperfekt) [DE]
 3 - Сгенерировать (Präsens) [DE]
 4 - Проверить своё (Präsens) [DE]
 5 - Сгенерировать (Modal + Infinitiv) [DE]
 6 - Проверить своё (Modal + Infinitiv) [DE]
 7 - Сгенерировать (Азербайджанский: ADV+OBJ Present) [AZ]
 8 - Проверить своё (Азербайджанский: ADV+OBJ Present) [AZ]
 0 - Выход
> 4
Введите предложение (Präsens):
>> Du machst oft, das_Buch.
✅ Всё верно!
```

Рис. 7: Modal + Infinitiv.

Верное предложение Азербайджанское:

```

Меню:
 1 – Сгенерировать (Plusquamperfekt) [DE]
 2 – Проверить своё (Plusquamperfekt) [DE]
 3 – Сгенерировать (Präsens) [DE]
 4 – Проверить своё (Präsens) [DE]
 5 – Сгенерировать (Modal + Infinitiv) [DE]
 6 – Проверить своё (Modal + Infinitiv) [DE]
 7 – Сгенерировать (Азербайджанский: ADV+OBJ Present) [AZ]
 8 – Проверить своё (Азербайджанский: ADV+OBJ Present) [AZ]
 0 – Выход
> 8
Введите предложение (Азербайджанский):
>> 0 dunen mashini, gögür?
✓ Всё верно!

```

Рис. 8: Проверка Azerbaijan.

Верное предложение Präsens:

```

Меню:
 1 – Сгенерировать (Plusquamperfekt) [DE]
 2 – Проверить своё (Plusquamperfekt) [DE]
 3 – Сгенерировать (Präsens) [DE]
 4 – Проверить своё (Präsens) [DE]
 5 – Сгенерировать (Modal + Infinitiv) [DE]
 6 – Проверить своё (Modal + Infinitiv) [DE]
 7 – Сгенерировать (Азербайджанский: ADV+OBJ Present) [AZ]
 8 – Проверить своё (Азербайджанский: ADV+OBJ Present) [AZ]
 0 – Выход
> 3
Сгенерировано: Es isst dich!

Меню:
 1 – Сгенерировать (Plusquamperfekt) [DE]
 2 – Проверить своё (Plusquamperfekt) [DE]
 3 – Сгенерировать (Präsens) [DE]
 4 – Проверить своё (Präsens) [DE]
 5 – Сгенерировать (Modal + Infinitiv) [DE]
 6 – Проверить своё (Modal + Infinitiv) [DE]
 7 – Сгенерировать (Азербайджанский: ADV+OBJ Present) [AZ]
 8 – Проверить своё (Азербайджанский: ADV+OBJ Present) [AZ]
 0 – Выход
> 4
Введите предложение (Präsens):
>> Es isst dich!
✓ Всё верно!

```

Рис. 9: Präsens + проверка.

Заключение

Разработана и формально описана **LL(1)**-грамматика для немецкого времени *Plusquamperfekt*. В коде на *Haskell* реализованы:

- генератор случайных предложений, строго подчинённых правилам грамматики;
- линейный распознаватель, проверяющий принадлежность строки языку.

Кроме основной цели, модуль поддерживает ещё три подмножества естественных языков:

- немецкое **Präsens** — согласование глаголов по лицу/числу;
- конструкцию **Modal + Infinitiv** (DE) с правильной формой модального глагола;
- азербайджанский **Present Perfect**-шаблон «SUBJ ADV OBJ VERB», где глагол также согласуется с подлежащим.

Все реализованные грамматики являются контекстно-свободными, так как во всех реализован ϵ -переход.

Граматики так же принадлежат $LL(1)$.

Плюсы реализации

- все четыре временные формы описаны одной архитектурой и разделяют вспомогательные функции;
- грамматики действительно $LL(1)$ — распознавание выполняется за $O(n)$;
- лексиконы вынесены в списки — легко дополнять словарь без изменения алгоритмов.

Минусы

- не моделируется выбор *sein/haben* в *Plusquamperfekt*;
- отсутствует род-числовое согласование детерминатива и существительного;
- генерация пока не учитывает семантические ограничения (некоторые AUX-Partizip комбинации некорректны).

Масштабирование

- добавить отрицательные и вопросительные формы (достаточно расширить правила и словари);
- учесть род и падеж с помощью системы признаков либо путём расширения лексиконов;
- внедрить таблицу выбора *sein/haben* по классу глагола для более точного *Plusquamperfekt*;
- обобщить механизм согласования и использовать его для новых языков — модуль уже содержит примеры для немецкого и азербайджанского.

Приложение А. Порождающая грамматика (Pluperfect)

1. $\langle S \rangle ::= \langle NP_{\text{subj}} \rangle \langle VP_1 \rangle$
2. $\langle NP_{\text{subj}} \rangle ::= \langle Pronoun \rangle$
3. $\langle VP_1 \rangle ::= \langle Aux \rangle \langle VP_2 \rangle$
4. $\langle Aux \rangle ::= \text{auxsPlu}$
5. $\langle VP_2 \rangle ::= \langle OptAdv \rangle \langle OptObj \rangle \langle V_{en} \rangle$
6. $\langle OptAdv \rangle ::= \langle Adv \rangle \mid \varepsilon$
7. $\langle OptObj \rangle ::= \langle NP_{\text{obj}} \rangle \mid \varepsilon$
8. $\langle NP_{\text{obj}} \rangle ::= \langle Det \rangle \langle N \rangle$
9. $\langle Det \rangle ::= \text{determinersAcc}$
10. $\langle N \rangle ::= \text{nounsAcc}$
11. $\langle Adv \rangle ::= \text{adverbs}$
12. $\langle V_{en} \rangle ::= \text{pparts}$
13. $\langle Pronoun \rangle ::= \text{pronouns}$