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

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, пренадлежашего 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* продукции с 7–8 примерами терминалов каждой части речи.

1.1 Plusquamperfekt (PQ)

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

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

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

1.2 Präsens (PR)

SUBJ VERB [ADV] OBJ

$$
\begin{aligned}
\langle S \rangle &::= \langle Pronoun \rangle\ \langle Verb \rangle\ \langle Tail \rangle \\
\langle Tail \rangle &::= \langle Obj \rangle \mid \langle Adverb \rangle\ \langle Comma \rangle\ \langle Obj \rangle \\
\langle Pronoun \rangle &::= \texttt{ich} \mid \texttt{du} \mid \texttt{er} \mid \texttt{sie} \mid \texttt{es} \mid \texttt{wir} \mid \texttt{ihr} \mid \texttt{Sie} \\
\langle Verb \rangle &::= \texttt{sehe} \mid \texttt{siehst} \mid \texttt{sieht} \mid \texttt{essen} \mid \texttt{isst} \mid \texttt{macht} \mid \texttt{geht} \mid \texttt{schläft} \\
\langle Adverb \rangle &::= \texttt{heute} \mid \texttt{gestern} \mid \texttt{schon} \mid \texttt{oft} \mid \texttt{morgen} \mid \texttt{selten} \mid \texttt{immer} \\
\langle Obj \rangle &::= \texttt{den\_Apfel} \mid \texttt{das\_Buch} \mid \texttt{die\_Katze} \mid \texttt{den\_Hund} \mid \texttt{mich} \mid \texttt{dich} \mid \texttt{ihn} \mid \texttt{nichts} \\
\langle Comma \rangle &::= \text{“,”}
\end{aligned}
$$

Каждое правило имеет форму $A \to \alpha \Rightarrow G_{PR}$ — КС.

1.3 Modal + Infinitiv (MI)

SUBJ MODAL ADV [OBJ] INF

$$
\begin{aligned}
\langle S \rangle &::= \langle Pronoun \rangle\ \langle Modal \rangle\ \langle Adverb \rangle\ \langle Rest \rangle \\
\langle Rest \rangle &::= \langle Comma \rangle\ \langle Inf \rangle \mid \langle Obj \rangle\ \langle Comma \rangle\ \langle Inf \rangle \\
\langle Pronoun \rangle &::= \texttt{ich} \mid \texttt{du} \mid \texttt{er} \mid \texttt{sie} \mid \texttt{es} \mid \texttt{wir} \mid \texttt{ihr} \mid \texttt{Sie} \\
\langle Modal \rangle &::= \texttt{will} \mid \texttt{willst} \mid \texttt{wollen} \mid \texttt{kann} \mid \texttt{kannst} \mid \texttt{müssen} \\
\langle Adverb \rangle &::= \texttt{heute} \mid \texttt{gestern} \mid \texttt{schon} \mid \texttt{oft} \mid \texttt{morgen} \mid \texttt{bald} \mid \texttt{gleich} \\
\langle Obj \rangle &::= \texttt{den\_Apfel} \mid \texttt{das\_Buch} \mid \texttt{die\_Katze} \mid \texttt{den\_Hund} \mid \texttt{mich} \mid \texttt{dich} \mid \texttt{ihn} \mid \texttt{nichts} \\
\langle Inf \rangle &::= \texttt{sehen} \mid \texttt{essen} \mid \texttt{machen} \mid \texttt{gehen} \mid \texttt{schlafen} \mid \texttt{schreiben} \mid \texttt{sagen} \mid \texttt{finden} \\
\langle Comma \rangle &::= \text{“,”}
\end{aligned}
$$

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

1.4 Azeri Present (AZ)

SUBJ ADV OBJ "VERB

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

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

1.5 Правило пунктуации

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

$$\boxed{\langle Comma \rangle \;\rightarrow\; \text{“,”}}$$

и включаемый в структуры где требуется:

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

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

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 $\mathrm{FIRST}(\alpha) \cap \mathrm{FIRST}(\beta) = \varnothing$ и, если $\beta \Rightarrow^* \epsilon$, $\mathrm{FIRST}(\alpha) \cap \mathrm{FOLLOW}(A) = \varnothing$. 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, $T \cap N = \varnothing$;
- $S \in N$ — стартовый символ;
- $R \subseteq N \times (T \cup N)^*$ — конечное множество правил вида $A \to \alpha$ $(A \in N)$.

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

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

3 Дерево разбора для Pluperfect

На рисунке 1, показано дерево разбора для грамматики pluperfect. Дерево строилось снизу вверх.

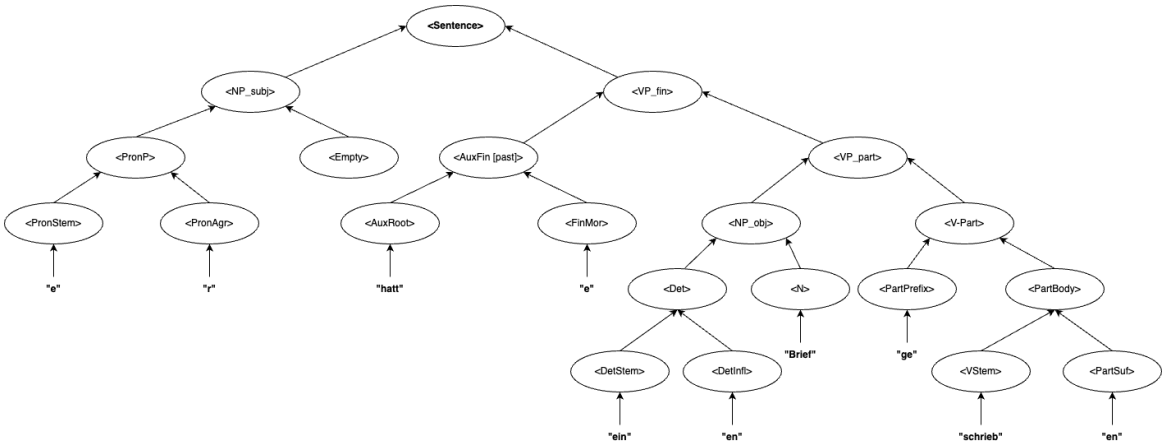

Рис. 1: Дерево разбора: Pluperfect

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

```
advM <- randMaybe advDE
objM <- randMaybe (,) <> detsAcc <*> nounsAcc
...
core = case (advM, objM) of                -- epsilon-vetka!!
  (Nothing, Nothing)        -> [subj, aux, pp]        -- <Mid> -> epsilon!!
  (Just a , Nothing)        -> [subj, aux, a, pp]
  ...

-- CHECK !!! <Mid> ::= epsilon | Adv | ObjGrp | Adv "," ObjGrp
validMid []                             = True            -- epsilon!!
validMid [adv]                          = adv `elem` advDE
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**

```
module Main where
import Lib
import System.IO (hFlush, stdout)

main :: IO ()
main = do
  putStr ""
  menuLoop

menuLoop :: IO ()
menuLoop = do
  putStr "> "
  hFlush stdout
  choice <- getLine
  case choice of
    "1" -> deriveAndShow deriveRandomPluperfect >> menuLoop
    "2" -> askAndCheck "Plusquamperfekt" isPluperfect >> menuLoop
    "3" -> deriveAndShow deriveRandomPresent >> menuLoop
    "4" -> askAndCheck "Prasens" isPresent >> menuLoop
    "5" -> deriveAndShow deriveRandomModal >> menuLoop
    "6" -> askAndCheck "Modal + Infinitiv" isModal >> menuLoop
    "7" -> deriveAndShow deriveRandomAzeri >> menuLoop
    "8" -> askAndCheck "Azerbaijan" isAzeri >> menuLoop
    "0" -> putStrLn "Sagolun / Aufiderzein!"
    ""  -> putStrLn "Sagolun / Aufiderzein!"
    _   -> putStrLn "Choose menu nums!!!" >> menuLoop
  where
```

```
    deriveAndShow gen = gen >>= putStrLn . ("Generated: " ++)
    askAndCheck tag predF = do
      putStrLn $ "Enter sentence (" ++ tag ++ "):"
      putStr " >> "
      hFlush stdout
      sent <- getLine
      putStrLn $ if predF sent then "Good" else "Not"
```

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

Lib.hs:

```
module Lib (
    deriveRandomPluperfect, deriveRandomPresent, deriveRandomModal,
       deriveRandomAzeri,
    isPluperfect, isPresent, isModal, isAzeri
  ) where

import System.Random (randomRIO)
import Data.Maybe    (isJust)
import Data.List     (find)

type Terminal = String

randFrom :: [a] -> IO a
randFrom xs = (xs !!) <$> randomRIO (0, length xs - 1)

randMaybe :: [a] -> IO (Maybe a)
randMaybe xs = do
  flag <- randomRIO (0 :: Int, 1)
  if flag == 1 then Just <$> randFrom xs else pure Nothing

maybeToList :: Maybe a -> [a]
maybeToList (Just x) = [x]
maybeToList Nothing  = []

pronouns :: [Terminal]
pronouns = ["ich","du","er","sie","es","wir","ihr","Sie"]

personIndexDE :: Terminal -> Int
personIndexDE p
  | p == "ich"                = 0
  | p == "du"                 = 1
  | p `elem` ["er","sie","es"] = 2
  | p == "wir"                = 3
  | p == "ihr"                = 4
  | p == "Sie"                = 5
```

```
  | otherwise                    = error "unknown pronoun"

adverbs :: [Terminal]
adverbs = ["heute","gestern","schon","oft","morgen"]

auxsPlu :: [Terminal]
auxsPlu = ["hatte","hattest","hatten","hattet","war","warst","waren","wart"]

pparts :: [Terminal]
pparts = ["gesehen","gegessen","gemacht","gegangen","geschlafen","
    geschrieben","gesagt"]

determinersAcc :: [Terminal]
determinersAcc = ["einen","eine","ein","den","die","das"]

nounsAcc :: [Terminal]
nounsAcc = ["Brief","Apfel","Hund","Katze","Buch","Auto","Haus","Zeitung"]

randMaybeObject :: IO (Maybe (Terminal, Terminal))
randMaybeObject = do
  flag <- randomRIO (0 :: Int, 1)
  if flag == 1 then do
    det  <- randFrom determinersAcc
    noun <- randFrom nounsAcc
    pure $ Just (det, noun)
  else
    pure Nothing

deriveRandomPluperfect :: IO String
deriveRandomPluperfect = do
  subj <- randFrom pronouns
  aux  <- randFrom auxsPlu
  advM <- randMaybe adverbs
  objM <- randMaybeObject
  ppar <- randFrom pparts
  let objTokens = maybe [] (\(d, n) -> [d, n]) objM
  pure . unwords $ [subj, aux] ++ maybeToList advM ++ objTokens ++ [ppar]

isPluperfect :: String -> Bool
isPluperfect str =
  case words str of
    (subj : aux : rest)
      | subj `elem` pronouns
      , aux  `elem` auxsPlu
      , not (null rest)
      , let ppart = last rest
      , ppart `elem` pparts
      , validMid (init rest)
      -> True
    _ -> False
  where
    validMid []                    = True
    validMid [adv]                 = adv `elem` adverbs
    validMid [det, noun]           = det `elem` determinersAcc && noun `
       elem` nounsAcc
    validMid [adv, det, noun]      = adv `elem` adverbs && det `elem`
       determinersAcc && noun `elem` nounsAcc
    validMid _                     = False

data GVerb = GVerb { gBase :: Terminal, gForms :: [Terminal] }

```

```haskell
verbsPresent :: [GVerb]
verbsPresent =
  [ GVerb "sehen"    ["sehe","siehst","sieht","sehen","seht","sehen"]
  , GVerb "essen"    ["esse","isst","isst","essen","esst","essen"]
  , GVerb "machen"   ["mache","machst","macht","machen","macht","machen"]
  , GVerb "gehen"    ["gehe","gehst","geht","gehen","geht","gehen"]
  , GVerb "schlafen" ["schlafe","schlAAfst","schlAAft","schlafen","schlaft",
      "schlafen"]
  ]

objects :: [Terminal]
objects = ["den_Apfel","das_Buch","die_Katze","den_Hund","mich","dich","ihn"
    ,"sie","es","nichts"]

conjugateDE :: Terminal -> GVerb -> Terminal
conjugateDE pron v = gForms v !! personIndexDE pron

deriveRandomPresent :: IO String
deriveRandomPresent = do
  subj <- randFrom pronouns
  v    <- randFrom verbsPresent
  advM <- randMaybe adverbs
  obj  <- randFrom objects
  let verb = conjugateDE subj v
  pure . unwords $ [subj, verb] ++ maybeToList advM ++ [obj]

isPresent :: String -> Bool
isPresent s =
  case words s of
    (subj : verb : rest)
      | subj `elem` pronouns
      , let idx = personIndexDE subj
      , isJust (find (\v -> gForms v !! idx == verb) verbsPresent)
      , validTail rest
      -> True
    _ -> False
  where
    validTail [obj]          = obj `elem` objects
    validTail [adv, obj]     = adv `elem` adverbs && obj `elem` objects
    validTail _              = False

data MVerb = MVerb { mBase :: Terminal, mForms :: [Terminal] }

modals :: [MVerb]
modals =
  [ MVerb "wollen"  ["will","willst","will","wollen","wollt","wollen"]
  , MVerb "konnen"  ["kann","kannst","kann","konnen","konnt","konnen"]
  , MVerb "mussen"  ["muss","musst","muss","mussen","musst","mussen"]
  ]

infinitives :: [Terminal]
infinitives = ["sehen","essen","machen","gehen","schlafen","schreiben","
    sagen"]

conjugateModal :: Terminal -> MVerb -> Terminal
conjugateModal p m = mForms m !! personIndexDE p

deriveRandomModal :: IO String
deriveRandomModal = do
  subj <- randFrom pronouns
  mverb <- randFrom modals
```

```
  adv  <- randFrom adverbs
  objM <- randMaybe objects
  inf  <- randFrom infinitives
  let modalForm = conjugateModal subj mverb
  pure . unwords $ [subj, modalForm, adv] ++ maybeToList objM ++ [inf]

isModal :: String -> Bool
isModal s =
  case words s of
    (subj : modal : adv : rest)
      | subj `elem` pronouns
      , adv  `elem` adverbs
      , let idx = personIndexDE subj
      , isJust (find (\m -> mForms m !! idx == modal) modals)
      , validTail rest
      -> True
    _ -> False
  where
    validTail [inf]          = inf `elem` infinitives
    validTail [obj, inf]     = obj `elem` objects && inf `elem` infinitives
    validTail _              = False

pronAze :: [Terminal]
pronAze = ["men","sen","o","biz","siz","onlar"]

personIndexAZ :: Terminal -> Int
personIndexAZ p = case p of
  "men"   -> 0
  "sen"   -> 1
  "o"     -> 2
  "biz"   -> 3
  "siz"   -> 4
  "onlar" -> 5
  _       -> error "unknown Azeri pronoun"

data AzVerb = AzVerb { azRoot :: Terminal, azForms :: [Terminal] }

azVerbs :: [AzVerb]
azVerbs =
  [ AzVerb "oxu" ["oxuyuram","oxuyursan","oxuyur","oxuyuruq","oxuyursunuz","oxuyurlar"]
  , AzVerb "yaz" ["yaziram","yazirsan","yazir","yaziriq","yazirsiniz","yazirlar"]
  , AzVerb "get" ["gedirem","gedirsen","gedir","gedirik","gedirsiniz","gedirler"]
  , AzVerb "gor" ["gorurem","gorursen","gorur","goruruk","gorursunuz","gorurler"]
  , AzVerb "al"  ["aliram","alirsan","alir","aliriq","alirsiniz","alirlar"]
  , AzVerb "sev" ["sevirem","sevirsen","sevir","sevirik","sevirsiniz","sevirler"]
  ]

adverbsAze :: [Terminal]
adverbsAze = ["bugun","dunen","tez-tez","hemishe"]

objectsAze :: [Terminal]
objectsAze = ["kitabi","evi","alma","mashini","mektubu"]

conjugateAZ :: Terminal -> AzVerb -> Terminal
conjugateAZ p v = azForms v !! personIndexAZ p
```

```haskell
deriveRandomAzeri :: IO String
deriveRandomAzeri = do
  subj <- randFrom pronAze
  adv  <- randFrom adverbsAze
  obj  <- randFrom objectsAze
  v    <- randFrom azVerbs
  let verb = conjugateAZ subj v
  pure . unwords $ [subj, adv, obj, verb]

isAzeri :: String -> Bool
isAzeri s =
  case words s of
    [subj, adv, obj, verb]
      | subj `elem` pronAze
      , adv  `elem` adverbsAze
      , obj  `elem` objectsAze
      , let idx = personIndexAZ subj
      , isJust (find (\v -> azForms v !! idx == verb) azVerbs)
      -> True
    _ -> 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 алфовита:

```
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 алфовита.

Ошибка - не в грамматике:

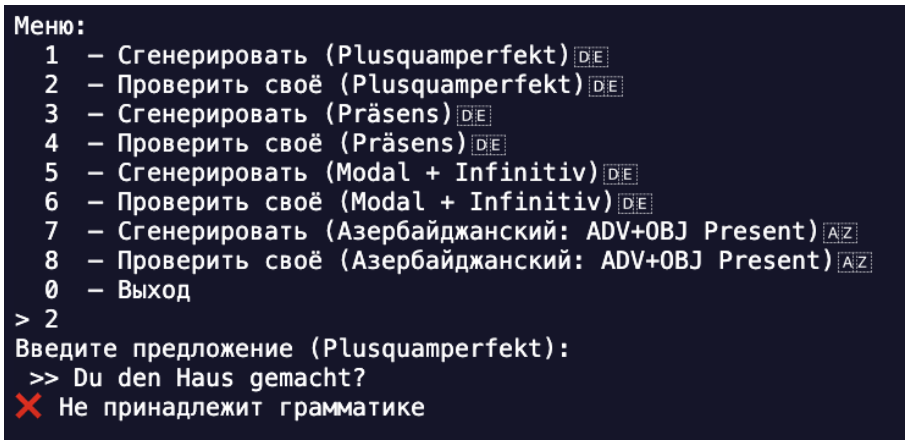

Рис. 5: Не пренадлежит грамматике.

Генерация предложения на азербайджанском Present:

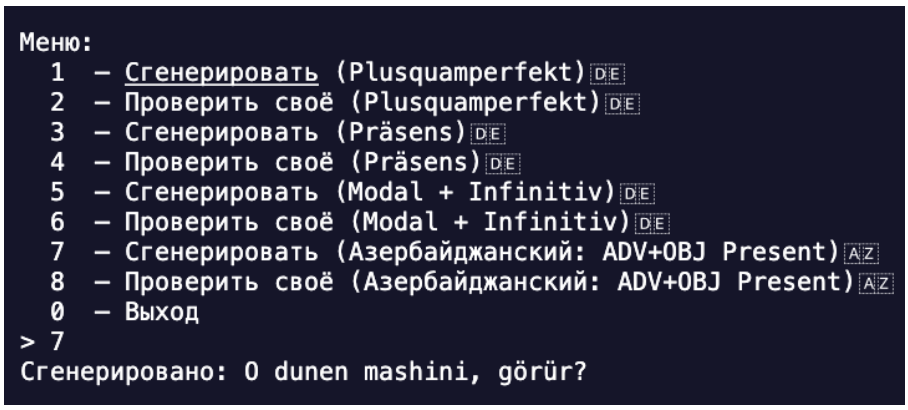

Рис. 6: Azerbaijan Present.

Неверное предложение Modal+Infinitiv:

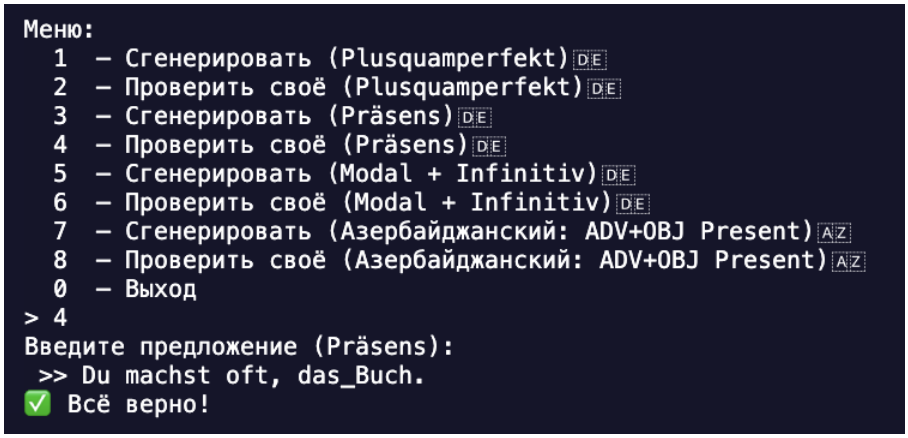

Рис. 7: Modal + Infinitiv.

Верное предложение Азербайджанское:

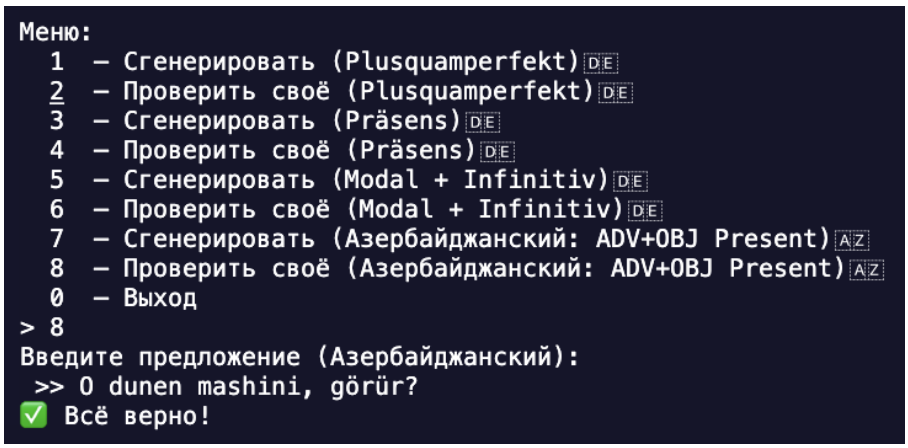

Рис. 8: Проверка Azerbaijan.

Верное предложение Prasens:

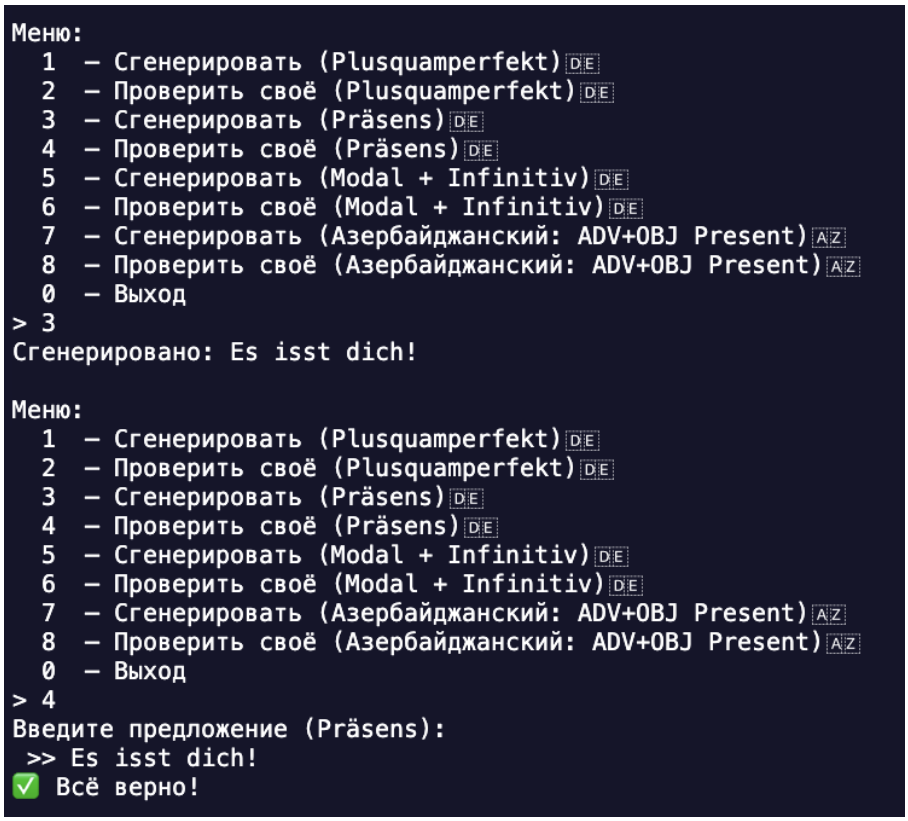

Рис. 9: Prasens + проверка.

Заключение

В данной работе была разработана и формально описана **LL(1)**-грамматика для подмножества естественного языка - немецкого времени *Plusquamperfekt*. Полученная грамматика является контекстно-свободной (КС), так как все правила имеют вид $A \to \alpha$, где $A \in N$ - нетерминал, а $\alpha \in (T \cup N)^*$ - цепочка из терминалов и нетерминалов. Наличие ϵ-переходов в правилах вида $A \to \varepsilon$ также подтверждает принадлежность грамматики к классу КС-грамматик.

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. Это означает, что порождаемые ими языки $L(G)$ также являются контекстно-свободными, где G - соответствующая грамматика. Формально это можно записать как:

$$L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$$

где S - начальный символ грамматики, T - множество терминалов, а $\Rightarrow^*$ - рефлексивно-транзитивное замыкание отношения вывода.

Грамматики также принадлежат классу $LL(1)$, что обеспечивает эффективный нисходящий разбор без возвратов.

Плюсы реализации

- все четыре временные формы описаны одной архитектурой и содержат 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. $\langle Aux \rangle ::=$ 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. $\langle Det \rangle ::=$ determinersAcc
10. $\langle N \rangle ::=$ nounsAcc
11. $\langle Adv \rangle ::=$ adverbs
12. $\langle V_{en} \rangle ::=$ pparts
13. $\langle Pronoun \rangle ::=$ pronouns