**Розділ 4**

**Варіант 15**

**Виконав Ходаков Максим, ТТП-32**

**Умова**

**Контекстно-вільні граматики. (До прописаних в умові балів для Haskell-реалізації слід додати 3 бали, а для Prolog-реалізації - кількість балів збільшується на 6).**

**15** Реалізація перевірки КВ-граматики бути граматикою Кореняка-Хопкрофта (у граматиці Кореняка-Хопкрофта для кожного з нетерміналів правила мають починатися з різних термінальних символів).

**Haskell:**

import Data.Char (isUpper, isSpace)  
import Data.List (groupBy, nubBy, sortBy)  
import Data.Function (on)  
  
-- Визначення базових типів  
type Symbol = String  
type Rule = (Symbol, String)  
type Grammar = [Rule]  
  
-- Функція для перевірки, чи є символ нетермінальним  
isNonTerminal :: Symbol -> Bool  
isNonTerminal = all (\c -> isUpper c || isSpace c)  
  
-- Функція для парсингу правил виводу з рядка  
parseRule :: String -> Maybe Rule  
parseRule input =  
 let (left, right) = break (== ' ') input  
 in if isNonTerminal left && not (null right) && length left == 1  
 then Just (left, dropWhile isSpace right)  
 else Nothing  
  
-- Функція для перевірки, чи є граматика Кореняка-Хопкрофта  
isChomskyHopcroft :: Grammar -> Bool  
isChomskyHopcroft grammar = all isUniqueFirstSymbols groupedRules  
 where  
 groupedRules = groupBy ((==) `on` fst) $ sortBy (compare `on` fst) grammar  
 isUniqueFirstSymbols rules = (length . nubBy ((==) `on` (take 1 . snd)) $ rules) == length rules  
  
-- Головна функція  
main :: IO ()  
main = do  
 putStrLn "Введіть правила граматики, кожне правило з нового рядка. Завершіть вводом пустого рядка."  
 inputLines <- getLines []  
 let parsedRules = mapM parseRule inputLines  
 case parsedRules of  
 Just grammar ->  
 if isChomskyHopcroft grammar  
 then putStrLn "Граматика є КВ граматикою Кореняка-Хопкрофта."  
 else putStrLn "Граматика НЕ є КВ граматикою Кореняка-Хопкрофта."  
 Nothing -> putStrLn "Кожне правило має мати один нетермінал ліворуч! Граматика не КВ!"  
  
-- Допоміжна функція для читання багаторядкового вводу  
getLines :: [String] -> IO [String]  
getLines lines = do  
 line <- getLine  
 if null line  
 then return lines  
 else getLines (lines ++ [line])

**Тести**

**A black screen with white text

Description automatically generated**

**A black screen with white text

Description automatically generated**

**A black screen with white text

Description automatically generated**

**A black screen with white text

Description automatically generated**

**A black screen with white text

Description automatically generated**

**Опис**

Ця програма на Haskell перевіряє, чи є дана граматика Контекстно -Вільною граматикою (КВ) Кореняка-Хопкрофта, виконуючи наступні кроки: Ініціалізація (Нульовий Крок)

Імпорт необхідних модулів: Data.Char, Data.List, Data.Function для роботи з текстом, списками та функціями відповідно.

Визначення типів даних: Встановлюється структура даних для зручної роботи з граматиками, правилами та символами. Валідація нетермінальних символів: Функція isNonTerminal перевіряє, чи символ є нетерміналом, враховуючи, що в КВГ ліва частина правила містить лише один нетермінал.

Загальний Крок Ітерації

Введення даних: Програма запрошує користувача ввести правила граматики, одне правило на рядок, завершуючи введенням пустого рядка. Парсинг правил: Функція parseRule аналізує кожен рядок введення, перетворюючи його на структуроване правило граматики або вказує на помилку, якщо правило не відповідає формату КВГ.

Перевірка на КВГ Кореняка-Хопкрофта: Функція isChomskyHopcroft перевіряє, чи відповідає граматика критеріям граматики Кореняка-Хопкрофта, а саме чи кожен нетермінал веде до унікального початкового термінала в своїх правилах виводу.

Умова Припинення Ітерації Ітераційний процес завершується, коли користувач вводить пустий рядок. Це сигнал для програми, що введення правил граматики завершено.

Оцінка Максимальної Кількості Кроків Максимальна кількість кроків залежить від кількості правил у введеній граматиці. Кожне правило обробляється один раз, тому максимальна кількість кроків лінійно залежить від кількості введених правил.

Завершуваність Алгоритму Алгоритм гарантовано завершується з кількома причин: Обмежена кількість вводу: Користувач вводить обмежену кількість правил, завершуючи пустим рядком. Перевірка на КВГ Кореняка-Хопкрофта виконується за один прохід: Після введення всіх правил, програма виконує одноразову перевірку цих правил, не вдаючись до нескінченних циклів. Структурована обробка даних: Кожен крок обробки (від вводу до парсингу і перевірки) чітко визначений і виконується послідовно, з певною метою на кожному етапі.

**Prolog:**

% Перевірка, чи всі правила з одним нетерміналом починаються з різних терміналів.

is\_korenyak\_grammar(Nonterminals) :-

check\_rules(Nonterminals).

% Для кожного нетерміналу перевірити правила.

check\_rules([]).

check\_rules([N | Rest]) :-

findall(FirstSymbol, (rule(N, [FirstSymbol | \_]), terminal(FirstSymbol)), FirstSymbols),

all\_unique(FirstSymbols),

check\_rules(Rest).

% Перевірка на унікальність перших символів.

all\_unique([]).

all\_unique([X | Xs]) :-

\+ member(X, Xs),

all\_unique(Xs).

% Функція для користувацького вводу.

iskorenyak(Nonterminals, Terminals, Rules) :-

assert\_nonterminals(Nonterminals),

assert\_terminals(Terminals),

assert\_rules(Rules),

is\_korenyak\_grammar(Nonterminals),

retract\_rules(Rules),

retract\_nonterminals(Nonterminals),

retract\_terminals(Terminals).

% Додавання нетерміналів до бази даних.

assert\_nonterminals([]).

assert\_nonterminals([N | Rest]) :-

assertz(nonterminal(N)),

assert\_nonterminals(Rest).

% Додавання терміналів до бази даних.

assert\_terminals([]).

assert\_terminals([T | Rest]) :-

assertz(terminal(T)),

assert\_terminals(Rest).

% Додавання правил до бази даних.

assert\_rules([]).

assert\_rules([N -> R | Rest]) :-

assertz(rule(N, R)),

assert\_rules(Rest).

% Видалення нетерміналів з бази даних.

retract\_nonterminals([]).

retract\_nonterminals([N | Rest]) :-

retract(nonterminal(N)),

retract\_nonterminals(Rest).

% Видалення терміналів з бази даних.

retract\_terminals([]).

retract\_terminals([T | Rest]) :-

retract(terminal(T)),

retract\_terminals(Rest).

% Видалення правил з бази даних.

retract\_rules([]).

retract\_rules([N -> R | Rest]) :-

retract(rule(N, R)),

retract\_rules(Rest).

**Тести**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Опис**

Описаний алгоритм в Prolog перевіряє, чи всі правила граматики, які містять лише один нетермінал у лівій частині, починаються з різних терміналів у правій частині. Це забезпечує відсутність невизначеностей при розборі ланцюжків мови, що генеруються цією граматикою. Описаний алгоритм можна поділити на наступні ключові етапи:

Ініціалізація (нульовий крок) Вхідні дані: Списки нетерміналів, терміналів та правил граматики. Додавання фактів: Вхідні списки трансформуються в базу даних Prolog шляхом додавання фактів через предикати assert\_nonterminals, assert\_terminals, та assert\_rules.

Загальний крок ітерації Перевірка правил для кожного нетермінала: Для кожного нетермінала зі списку виконується пошук перших символів правил, які є терміналами, та зберігаються у списку FirstSymbols. Перевірка на унікальність: Для кожного зібраного списку FirstSymbols перевіряється, чи всі елементи унікальні за допомогою all\_unique.

Умова припинення ітерації Ітерація припиняється, коли перевірено всі нетермінали зі списку. Якщо для кожного нетермінала перші символи правил, які є терміналами, унікальні, то граматика відповідає вимогам.

Оцінка максимальної кількості кроків Максимальна кількість кроків визначається кількістю нетерміналів у списку Nonterminals. Для кожного нетермінала виконується не більше N операцій, де N — кількість правил, асоційованих з цим нетерміналом.

Обґрунтування завершуваності Алгоритм є завершуваним з наступних причин: Кінцевість даних: Вхідні дані (списки нетерміналів, терміналів, правил) є кінцевими. Обмеженість ітерацій: Кожен крок ітерації зменшує кількість необроблених нетерміналів, прямуючи до пустого списку.

**Висновок**

Було розроблено, протестовано та описано програми на Haskell та Prolog для перевірки КВ-граматики на відповідність критерію граматики Кореняка-Хопкрофта.