**Розділ 3**

**Варіант 4**

**Виконав Ходаков Максим, ТТП-32**

**Умова**

**Скінчені автомати. (До прописаних в умові балів для Haskell-**

**реалізації слід додати 3 бали, а для Prolog-реалізації - кількість балів збільшується**

**4** Для заданого слова w виявити, чи допускає скінчений автомат хоча б одне слово, що може бути подане у вигляді xw для деякого слова x. При ствердній відповіді навести приклад відповідного слова xw.

**Haskell:**

import Data.List (nub, find)  
import qualified Data.Map as Map  
import System.IO  
import Data.Maybe (fromMaybe, isJust)  
  
type State = Int  
type Symbol = Char  
type Transition = ((State, Symbol), State)  
type Automaton = (Map.Map (State, Symbol) State, State, [State])  
  
addTransition :: Automaton -> Transition -> Automaton  
addTransition (transitions, start, finals) ((from, symbol), to) =  
 (Map.insert (from, symbol) to transitions, start, finals)  
  
accepts :: Automaton -> String -> Bool  
accepts (transitions, start, finals) word =  
 let step (Just state) symbol = Map.lookup (state, symbol) transitions  
 step Nothing \_ = Nothing  
 finalState = foldl step (Just start) word  
 in case finalState of  
 Just state -> state `elem` finals  
 Nothing -> False  
  
alphabetFromAutomaton :: Automaton -> [Symbol]  
alphabetFromAutomaton (transitions, \_, \_) = nub [symbol | ((\_, symbol), \_) <- Map.toList transitions]  
  
readAutomaton :: FilePath -> IO Automaton  
readAutomaton path = do  
 contents <- readFile path  
 let (transitionLines, rest) = splitAt (length (lines contents) - 2) (lines contents)  
 transitions = map parseTransition transitionLines  
 startState = read $ head rest  
 finalStates = map read $ words $ last rest  
 transitionMap = Map.fromList transitions  
 return (transitionMap, startState, finalStates)  
  
parseTransition :: String -> Transition  
parseTransition line =  
 let [from, to, [symbol]] = words line  
 in ((read from, symbol), read to)  
  
tryPrefixes :: Automaton -> String -> [Symbol] -> String -> Int -> Maybe String  
tryPrefixes automaton word alphabet prefix depth  
 | accepts automaton (prefix ++ word) = Just prefix  
 | depth >= maxDepth = Nothing -- Максимальна глибина рекурсії досягнута  
 | otherwise = case filter isJust results of  
 [] -> Nothing  
 (Just p:\_) -> Just p  
 where  
 maxDepth = 10 -- Встановлюємо максимальну глибину рекурсії  
 possiblePrefixes = [prefix ++ [a] | a <- alphabet]  
 results = map (\p -> tryPrefixes automaton word alphabet p (depth + 1)) possiblePrefixes  
  
main :: IO ()  
main = do  
 putStrLn "Enter the path to the automaton file:"  
 filePath <- getLine  
 automaton <- readAutomaton filePath  
 putStrLn "Enter the word w:"  
 word <- getLine  
 let alphabet = alphabetFromAutomaton automaton  
 -- Початковий виклик функції tryPrefixes з глибиною рекурсії 0  
 let prefix = tryPrefixes automaton word alphabet "" 0  
 case prefix of  
 Just p -> putStrLn $ "The automaton accepts the word with prefix: " ++ p ++ word  
 Nothing -> putStrLn "No prefix found that the automaton accepts."

**Тести**

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**Опис**

Принцип роботи алгоритму полягає у визначенні, чи може скінченний автомат прийняти рядок, який є конкатенацією певного префіксу x та вхідного слова w. Для цього алгоритм спробує знайти такий префікс x, починаючи з порожнього рядка та поступово додаючи до нього символи з алфавіту автомата, доки не буде знайдено префікс, який, будучи доданим до w, дозволить автомату перейти в один із своїх фінальних станів.

Типи даних: Визначено основні типи для станів (State), символів (Symbol), переходів (Transition) та самого автомата (Automaton). Функція addTransition: Додає новий перехід до автомата. Функція accepts: Перевіряє, чи автомат приймає задане слово, використовуючи початковий стан та переходи для обробки кожного символа слова. Функція alphabetFromAutomaton: Витягує алфавіт (унікальні символи) з автомата. Функція readAutomaton: Зчитує автомат з файлу, визначаючи переходи, початковий стан та кінцеві стани. Функція tryPrefixes: Головна функція, що імплементує алгоритм пошуку префіксу. Вона рекурсивно генерує можливі префікси до певної глибини, перевіряючи, чи приймає автомат комбіноване слово (префікс + задане слово).

Ініціалізація (Нульовий крок): Починаємо з порожнього префіксу x та встановлюємо глибину рекурсії depth на 0. Визначаємо максимальну глибину рекурсії maxDepth, яка слугує умовою зупинки для алгоритму.

Загальний крок ітерації: Для поточного префіксу x, перевіряємо, чи приймає автомат рядок x ++ w. Якщо так, повертаємо x як валідний префікс. Якщо автомат не приймає x ++ w, генеруємо новий набір можливих префіксів, додаючи до поточного префіксу x кожен символ з алфавіту автомата. Рекурсивно викликаємо функцію tryPrefixes для кожного з нових можливих префіксів, збільшуючи глибину рекурсії на 1.

Умова припинення ітерації: Якщо глибина рекурсії depth досягає максимальної глибини maxDepth, припиняємо ітерацію та повертаємо Nothing, оскільки алгоритм не зміг знайти валідний префікс до досягнення цієї глибини.

Оцінка максимальної кількості кроків: Максимальна кількість кроків ітерації залежить від встановленої максимальної глибини рекурсії maxDepth та розміру алфавіту |Alphabet|. У найгіршому випадку, кількість кроків оцінюється як O(|Alphabet|^maxDepth), оскільки на кожному рівні рекурсії алгоритм генерує нові префікси, додаючи кожен символ з алфавіту.

**Prolog**

% Дозволяє динамічно змінювати базу знань  
:- dynamic transition/3.  
:- dynamic accepting/1.  
:- dynamic initial\_state/1.  
  
% Встановлення початкового стану  
set\_initial\_state(State) :- assertz(initial\_state(State)).  
  
% Додає факти про автомат  
add\_transition(From, Symbol, To) :- assertz(transition(From, Symbol, To)).  
add\_accepting(State) :- assertz(accepting(State)).  
  
% Видаляє всі факти про автомат (для нового вводу)  
clear\_automaton :- retractall(transition(\_, \_, \_)), retractall(accepting(\_)).  
  
% Модифікуємо правило move та accept, щоб використовувати заданий початковий стан  
move(State, [Char|Tail], NextState) :-  
 transition(State, Char, State2),  
 move(State2, Tail, NextState).  
move(State, [], State).  
  
accept(Word, State) :-  
 initial\_state(InitialState),  
 move(InitialState, Word, State),  
 accepting(State).  
  
% Генерує префікси, щоб слово було прийнято  
generate\_prefix(Word, Prefix) :-  
 findall(X, transition(\_, X, \_), Symbols), % Зібрати всі можливі символи  
 generate\_prefix\_iter(Word, Symbols, [], 10, Prefix).  
  
generate\_prefix\_iter([], \_, Prefix, \_, Prefix). % Базовий випадок: не залишилося символів для додавання  
  
generate\_prefix\_iter([Char|Tail], Symbols, Accumulated, Depth, Prefix) :-  
 Depth > 0,  
 ( accept([Char|Tail], \_) ->  
 reverse(Accumulated, Prefix) % Якщо слово приймається, повернути накопичений префікс  
 ; member(Symbol, Symbols),  
 append(Accumulated, [Symbol], NewAccumulated),  
 append([Symbol], [Char|Tail], NewWord), % Спробувати додати кожен символ до слова  
 NewDepth is Depth - 1,  
 generate\_prefix\_iter(NewWord, Symbols, NewAccumulated, NewDepth, Prefix)  
 ).

**Тести**

**A screenshot of a computer

Description automatically generated**

**A computer screen shot of a black screen

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Опис**

Даний код реалізує функцію generate\_prefix, яка генерує префікси для заданого слова так, щоб це слово було прийнято скінченим автоматом. Ось детальний опис роботи програми та алгоритму:

Опис роботи програми: Додавання фактів про автомат: Користувач може встановити початковий стан, додати переходи та вказати фінальні стани за допомогою предикатів set\_initial\_state, add\_transition і add\_accepting відповідно. Видалення фактів про автомат: Якщо потрібно почати з новим автоматом, можна видалити всі наявні переходи та фінальні стани за допомогою предиката clear\_automaton. Генерація префіксів: Функція generate\_prefix приймає слово Word та повертає префікси цього слова, які приймає автомат. Вона використовує ітеративний процес generate\_prefix\_iter, який рекурсивно додає символи до слова, перевіряє, чи приймається це слово автоматом, і повертає накопичений префікс у разі прийняття.

Опис алгоритму: Ініціалізація (нульовий крок): Початковий стан автомату встановлюється за допомогою set\_initial\_state. Функція generate\_prefix отримує на вхід слово Word та створює пустий префікс. Загальний крок ітерації: Функція generate\_prefix\_iter рекурсивно перебирає всі можливі комбінації символів, що можуть бути додані до слова Word, починаючи з пустого слова. Для кожного можливого символу додається до слова Word, і функція перевіряє, чи приймається нове слово автоматом за допомогою accept. Якщо слово приймається, префікс зберігається. Умова припинення ітерації: Ітерація завершується, коли вже не можливо додати нові символи до слова або коли досягнута максимальна глибина рекурсії (Depth = 0). Оцінка максимальної кількості кроків: Максимальна кількість кроків обмежується параметром Depth, який передається в generate\_prefix\_iter. У цьому випадку встановлено Depth = 10.

**Висновок**

Були побудовані програма на Haskell та Prolog, протестовані та описані принципи програмної реалізації програм.