Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Formálne jazyky
Dokumentácia k obhajobe zadania

Oleniuk Vladyslav
Akademický rok: 2024/2025
Študijný program: Informatika

Obsah

1 Formulácia zadania **2**
1.1 Vstup 2
1.2 Výstup 2

2 Vypracovanie zadania **2**
2.1 Opis architektúry riešenia 2
2.1.1 Lexer.py 2
2.1.2 Parser.py 3
2.1.3 main.py 3
2.2 Gramatika jazyka regulárnych výrazov 3
2.3 Podrobný postup implementácie 3
2.4 Ukážka vstupu a prezentácia výsledkov 5
2.5 Problémy a ich riešenie 5
2.6 Hodnotenie zadania 6

3 Vyhodnotenie a záver **6**

1 Formulácia zadania

Cieľom zadania je implementovať jednoduchý generátor konečnostavových automatov (NKA) zo zadaného regulárneho výrazu využitím metódy jeho syntaktickej analýzy zhora nadol rekurzívnym zostupom.

1.1 Vstup

Na vstupe sa očakáva ľubovoľný reťazec znakov reprezentujúci regulárny výraz v na tomto predmete štandardne využívanej notácii.

1.2 Výstup

Program bude mať nasledujúce výstupy:

- grafická reprezentácia stromu odvodenia pre zadaný regulárny výraz,
- plne funkčná programová implementácia nedeterministického konečnostavového akceptora zodpovedajúceho zadanému regulárnemu výrazu.

2 Vypracovanie zadania

Zadanie bolo vypracované v jazyku `Python`.

2.1 Opis architektúry riešenia

Všeobecná štruktúra programu je nasledovná:

```
src/
├── Lexer.py
├── Parser.py
├── main.py
└── (nka.py)
```

2.1.1 Lexer.py

`Lexer.py` rozdeľuje vstupný reťazec na lexémy a zakóduje ich do tokenov. Vytvoril som nasledujúce typy tokenov:

```
SYMBOL -> akykolvek iny symbol okrem nizsie uvedenych
LPAREN -> (
RPAREN -> )
```

```
LCBRA  -> {
RCBRA  -> }
LBRACK -> [
RBRACK -> ]
PIPE   -> |
EOF    -> koniec vstupu
```

2.1.2 Parser.py

Súbor `Parser.py` slúži na syntaktickú analýzu, pričom využíva techniku rekurzívneho zostupu. Počas analýzy konštruuje strom odvodenia.

2.1.3 main.py

Hlavný súbor aplikácie, prostredníctvom ktorého sa spúšťa. Zároveň vykonáva zápis výsledného NKA do súboru `nka.py`.

2.2 Gramatika jazyka regulárnych výrazov

```
    regular ::= 'eps' | alternative
alternative ::= sequence {'|' sequence}
   sequence ::= element {element}
    element ::= symbol | '(' alternative ')' |
                '[' alternative ']' | '{' alternative '}'
```

2.3 Podrobný postup implementácie

Najprv som definoval gramatiku regulárnych výrazov, ktorá je opísaná v časti 2.2. Potom som implementoval iteratívny a neiteratívny NKA pre binárne numerály, aby som zistil, čo bude jednoduchšie implementovať pomocou programu. Keďže iteratívny prístup vyžadoval len dve dynamicky generované polia 1 (tabuľka prechodov a akceptačné stavy), rozhodol som sa zostať pri tejto možnosti.

Listing 1: Dynamicky generované polia

```
_transition_table = {
0: {'': {1, 9}},
1: {'1': {2}},
2: {'': {3}},
3: {'': {4}},
```

```
4: {'': {5, 7}},
5: {'1': {6}},
6: {'': {3}},
7: {'0': {8}},
8: {'': {3}},
9: {'0': {10}},
10: {},
}
_accepted_states = {
8, 10, 3, 6,
}
```

Listing 2: StateNode

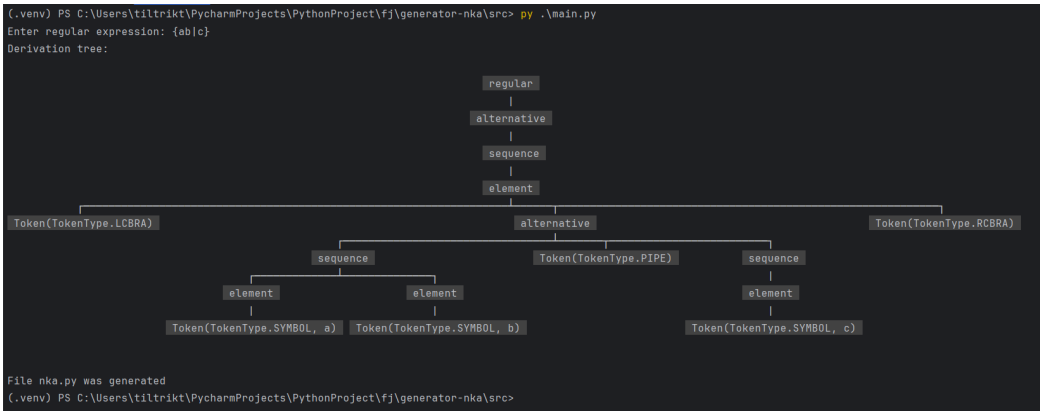
```
class StateNode:
def init(self):
self.transitions = {}
self.id = None


def add_transition(self, symbol, destination_node):
    if symbol not in self.transitions:
        self.transitions[symbol] = set()
    self.transitions[symbol].add(destination_node)
```

Ďalej som vytvoril kostru projektu pozostávajúcu zo súborov `Parser.py`, `Lexer.py` a `main.py`. Tieto súbori sú podrobnejšie opísané v časti 2.1. Pri súbori `Lexer.py` som sa nestretol so žiadnymi problémami.

V súbori `Parser.py` som najprv implementoval kostru pre všetky neterminálne symboly. Potom som pomocou Thompsonovho konštrukčného algoritmu vytvoril NKA a zároveň som pomocou knižnice `PrettyPrintTree` nakreslil strom odvodenia. Konečný NKA obsahoval informácie o počiatočnom stave a akceptačných stavoch. Informácie o prechodoch boli uložené v uzloch *StateNode* 2.

Výsledny NKA a stavy bolo potrebné previesť do tabuľky, ako je uvedené vyššie 1. To sa vykoná v súbori `main.py`. Tento súbor prijíma vstup od používateľa a volá `Parser.py`. Výsledok prevedie na tabuľku vo forme reťazca a pridá kód šablóny používanej pre iteratívny NKA. Výsledný reťazec sa zapíše do súboru `nka.py`.



Obr. 1: Príklad vstupu

2.4 Ukážka vstupu a prezentácia výsledkov

Po spustení programu sa vygeneruje strom zobrazený na obrázku 1 a súbor `nka.py`. Vygenerovaný súbor *nka.py* funguje nasledovne:

```
(.venv) PS ...\generator-nka\src> py .\nka.py
Enter a word:
Word '' is ACCEPTED!
Enter a word: ab
Word 'ab' is ACCEPTED!
Enter a word: c
Word 'c' is ACCEPTED!
Enter a word: ababcccabc
Word 'ababcccabc' is ACCEPTED!
Enter a word: acb
Word 'acb' is NOT ACCEPTED!
Enter a word: ab123
Word 'ab123' is NOT ACCEPTED!
Enter a word: quit
(.venv) PS ...\generator-nka\src>
```

2.5 Problémy a ich riešenie

Dlho mi trvalo, kým som pochopil, ako generovať tabuľku prechodov. Pokúsil som sa ju implementovať pomocou Thompsonovho konštrukčného algoritmu. Týmto spôsobom som vytvoril menšie NKA, ktoré som následne spájal do jedného väčšieho automatu podľa definovaných pravidiel. Po niekoľkých neúspešných

pokusoch sa mi konečne podarilo implementovať generovanie tabuľky prechodov pre jednoduchý regulárny výraz *ab*. Preto som sa rozhodol pokračovať v tomto prístupe a rozširovať jeho funkčnosť pre zložitejšie regulárne výrazy.

2.6 Hodnotenie zadania

Úloha bola veľmi zaujímavá a pomohla mi lepšie pochopiť mnohé témy. Jediným problémom, s ktorým som sa stretol, bolo nájsť knižnicu na nakreslenie stromu. Ušetril by som veľa času, keby bolo niekoľko poskytnutých v samotnom zadaní.

3 Vyhodnotenie a záver

Podarilo sa mi implementovať všetko. Úloha sa testovala na niekoľkých rôznych vstupoch: správnych, nesprávnych a prázdnych. Nemusel som sa zaseknúť na žiadnom bode na dlhší čas.