Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Formálne jazyky
Dokumentácia k obhajobe zadania

Lukáš Čeč
Akademický rok: 2024/2025
Študijný program: Informatika

Obsah

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. Pri vypracovaní zadania nie je povolené použitie žiadneho existujúceho generátora jazykových procesorov.

2 Vypracovanie zadania

2.1 Opis architektúry riešenia

Navrhnuté riešenie je rozdelené na backendovú a frontendovú časť. Backend je implementovaný v jazyku Python pomocou frameworku FastAPI. Jeho úlohou je prijímať regulárny výraz vo forme reťazca, parsovať ho pomocou rekurzívne zostupného syntaktického analyzátora, generovať derivančný strom a následne zo stromu vytvárať nedeterministický konečný automat (NKA). Výsledný automat je možné testovať na vstupe.

Frontend je implementovaný v Next.js. Slúži na vizualizáciu derivačného stromu a interaktívne zadávanie regulárnych výrazov. Používateľ môže vkladať výrazy, zobraziť ich derivančné stromy a skúšať, či konkrétne slová daný automat akceptuje. Súčasťou rozhrania je aj história zadaných výrazov.

Komunikácia medzi frontendom a backendom prebieha cez REST API. Každý regulárny výraz je na server odoslaný pomocou HTTP POST požiadavky, pričom odpoveď obsahuje serializovaný derivačný strom alebo výsledok simulácie automatu.

Celé riešenie je kontajnerizované pomocou Dockeru, aby bolo jednoducho spustiteľné na akomkoľvek systéme.

2.2 Gramatika jazyka regulárnych výrazov (EBNF)

Použitá gramatika jazyka regulárnych výrazov je nasledovná (v EBNF):

```
<regular>     ::= <alternative>

<alternative> ::= <sequence> { "|" <sequence> }

<sequence>    ::= { <element> }

<element>     ::= "{" <regular> "}"      // Iterácia (Kleeneho hviezda)
```

```
                | "[" <regular> "]"      // Voliteľnosť (alebo eps)
                | <symbol>
                | "eps"                    // Epsilon

<symbol>      ::= znak okrem špeciálnych znakov {, }, [, ], |
```

Vysvetlenie:

- **Tranzitívny uzáver** {} označujú iteráciu (Kleeneho hviezda, t.j. ľubovoľne veľa opakovaní).
- **Voliteľnosť** [ ] označujú voliteľnosť (t.j. výraz v zátvorkách alebo nič).
- **Alternatíva** | (pipe) vyjadruje alternatívu (alebo).
- **Epsilon** značí prázdne slovo.
- **Symbol** je ľubovoľný znak okrem vyššie uvedených špeciálnych znakov.

2.3 Podrobný postup implementácie

Implementácia sa skladá z niekoľkých hlavných častí:

1. **Lexer** – Tokenizuje vstupný reťazec regulárneho výrazu na sekvenciu tokenov (zátvorky, symboly, operátory).
2. **Parser** – Implementuje rekurzívne zostupný syntaktický analyzátor podľa EBNF gramatiky. Výsledkom je derivačný strom, v ktorom uzly reprezentujú pravidlá gramatiky a listy konkrétne symboly.
3. **Konštrukcia NKA** – Prechádza derivačný strom a podľa typu uzla (sekvencia, alternatíva, iterácia, voliteľnosť) generuje príslušnú štruktúru NKA pomocou známych konštrukčných pravidiel (napr. Thompsonova konštrukcia).
4. **Transofrmácia stromu** – Pre vizualizáciu na frontende sa strom transformuje do JSON formátu, kde každý uzol obsahuje svoj popis a deti.
5. **Simulátor NKA** – Slúži na overenie, či dané slovo je akceptované vytvoreným automatom, využíva uzávierku epsilon-pravidiel.
6. **Frontend vizualizácia** – Pomocou React Flow sa na základe JSON stromu vykreslí interaktívny graf derivačného stromu.

7. **História výrazov a testovanie slov** – Frontend si udržiava históriu výrazov a umožňuje testovať slová na prijatie automatom.

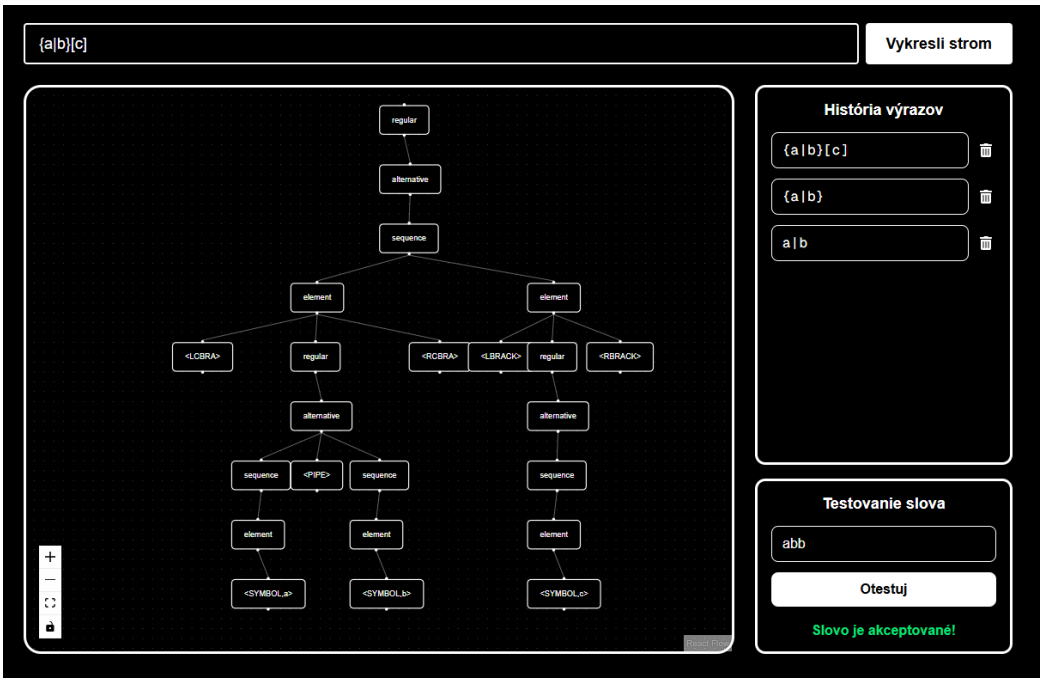
8. **Dockerizácia** – Backend aj frontend sú zabalené v separátnych Docker kontajneroch, spúšťané pomocou docker-compose.

2.4 Ukážka vstupu a prezentácia výsledkov

Príklad regulárneho výrazu:

`{a|b}[c]`

Derivačný strom (vizuálne):



Obr. 1: Ukážka derivačného stromu

Príklad simulácie:

- `regex = {a|b}[c]`, slovo `ab` → akceptované
- `regex = {a|b}[c]`, slovo `ac` → akceptované
- `regex = {a|b}[c]`, slovo `a` → akceptované

- `regex = {a|b}[c]`, slovo `abc` → akceptované
- `regex = {a|b}[c]`, slovo `bc` → akceptované
- `regex = {a|b}[c]`, slovo `b` → akceptované
- `regex = {a|b}[c]`, slovo `d` → neakceptované

Na obrázku je možné vidieť zobrazenie derivačného stromu pre konkrétny regulárny výraz. Používateľ môže ľubovoľne zadávať vlastné výrazy, vidieť ich stromy a testovať ľubovoľné slová.

2.5 Problémy a ich riešenie

Pri implementácii sa vyskytli nasledovné problémy:

- **Správne parsovanie zátvoriek a operátorov**: Bolo potrebné detailne ošetriť, aby sa zátvorky správne párovali a výrazy ako {ab—c} boli správne rozpoznané a rozdelené.
- **Vizualizácia stromu**: Bolo potrebné riešiť rozmiestnenie uzlov tak, aby bol strom čitateľný aj pri väčších výrazoch.

Všetky tieto problémy boli postupne riešené ladením, dôsledným testovaním rôznych prípadov a prečítaním odporúčanej literatúry k téme.

2.6 Hodnotenie zadania

Zadanie hodnotím ako veľmi zaujímavé, pretože som mohol priamo aplikovať teoretické poznatky z formálnych jazykov, tým že som si zadanie obohatil o prepojenie backendu s frontendom a vizualizáciou odovzdávaných dát. Práca s Dockerom mi umožnila pripraviť riešenie jednoducho spustiteľné na ľubovoľnom systéme.

3 Vyhodnotenie a záver

V zadaní sa podarilo úspešne implementovať generátor NKA zo zadaného regulárneho výrazu pomocou rekurzívne zostupného analyzátora. Riešenie vďaka webovej vizualizácii umožňuje jednoduché testovanie rôznych výrazov a slov. Celý systém je dockerizovaný a pripravený na jednoduché nasadenie. Ako možnosť rozšírenia do budúcnosti vidím napríklad podporu deterministických automatov, generovanie minimalizovaných automatov, podporu

ďalších operácií v regulárnych výrazoch, alebo export automatu do grafického formátu.

Priložené súbory:
README.txt – EBNF gramatika
Zdrojové kódy (backend, frontend)
Dockerfile, docker-compose.yml