VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SIMULACE LÍNÝCH KONEČNÝCH AUTOMATŮ

SIMULATION OF LAZY FINITE AUTOMATA

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE LUKÁŠ VRÁBEL

AUTHOR

VEDOUCÍ PRÁCE Ing. ROMAN LUKÁŠ, Ph.D.

SUPERVISOR

BRNO 2007

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Vrábel Lukáš**
Obor: Informační technologie
Téma: **Simulace líných konečných automatů**
Kategorie: Překladače

Pokyny:

1. Seznamte se podrobně s problematikou líných konečných automatů.
2. Navrhněte algoritmus, který efektivně odsimuluje činnost líného konečného automatu.
3. Vytvořte program, jehož vstupem bude vhodně popsaný líný konečný automat a jeho výstupem vygenerovaný program v jazyce C, který po kompilaci a spuštění odsimuluje činnost líného konečného automatu ze vstupu.
4. Zhodnoťte dosažené výsledky a diskutujte další možný vývoj projektu.

Literatura:

- Meduna, A.: Automata and Languages, Springer, London, 2000

Při obhajobě semestrální části projektu je požadováno:

- Body 1) a 2)

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Lukáš Roman, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Lukáš Vrábel**
Id studenta: 84101
Bytem: SNP 997/14, 924 01 Galanta
Narozen: 09. 09. 1985, Galanta
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   bakalářská práce

Název VŠKP: Simulace líných konečných automatů
Vedoucí/školitel VŠKP: Lukáš Roman, Ing., Ph.D.
Ústav: Ústav informačních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

Táto bakalárska práca popisuje matematický model lenivého konečného automatu a implementáciu jednoduchého simulačného programu postaveného na základoch tohoto automatu, ktorý demonštruje možnosti daného modelu. Taktiež sa zaoberá výhodami a nevýhodami oproti ostatným prístupom a implementáciám. Model lenivého konečného automatu poskytuje určité výhody z hladiska uživateľa, ktorý vďaka nemu dostáva do rúk väčšiu voľnosť pri definovaní pravidiel automatu.

Kľúčové slová

teória automatov, konečný automat, lenivý konečný automat, formálna gramatika, syntaktická analýza

Abstract

This bachelor's thesis introduces a formal model of the lazy finite state machine and implementation of application based on this model. The application demonstrates functionality of new model, and pros and cons of this approach. This model has advantage over existing models in easier definitions of transition rules.

Keywords

automata theory, finite state machine, lazy automata, formal grammar, syntactical analysis

Citácia

Lukáš Vrábel: Simulace líných konečných automatů, bakalářská práce, Brno, FIT VUT v Brně, 2007

Simulace líných konečných automatů

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Romana Lukáša, Ph.D.

. . . . . .
Lukáš Vrábel
15. května 2007

Poďakovanie

Na tomto mieste by som chcel poďakovať Ing. Romanovi Lukášovi, Ph.D. za pomoc, poskytnuté materiály a čas, ktorý mi venoval pri tvorbe tejto práce.

Obsah

Kapitola 1

Úvod

Táto bakalárska práca popisuje matematický model lenivého konečného automatu a implementáciu jednoduchého syntaktického analyzátoru postaveného na základoch tohoto automatu. Tento model je výhodný z hladiska uživateľa, ktorý vďaka nemu dostáva do rúk väčšiu voľnosť pri definovaní pravidiel automatu.

V druhej kapitole sú vysvetlené základné pojmy a matematické definície potrebné pre zvládnutie tematiky konečných automatov.

Za nimy nasleduje tretia kapitola, ktorá obsahuje krátky prehľad histórie automatov a vedeckých prác, ktoré napomohli ich rozvoju.

Nasledujúca kapitola popisuje prehľad základných typov konečných automatov a ich zadelenie do kategórií podľa rôznych kritérií. Sú tu spomenuté formálne definície, základné metódy na implementáciu, a porovnanie výhod a nevýhod rozličných prístupov.

Piata kapitola obsahuje popis samotného lenivého automatu, jeho formálny model a základné vlastnosti. Po nej nasleduje popis implemenácie a uživateľský manuál samotného programu simulujúceho lenivý konečný automat, ktorý tak demonštruje jeho možnosti v praxi.

V závere je zhrnutie celej práce, úvaha o možných rozšíreniach projektu, jeho prínos a zamyslenie sa nad ďalšími možnými využitiami nového modelu.

Kapitola 2

Konečný automat

Táto kapitola je úvodom ku konečným automatom, ktoré tvoria základ pre lenivé konečné automaty. Na začiatku je neformálny popis konečného automatu ako aj formálna definícia, po ktorej nasleduje použitie automatov, a spôsob ich zápisu. Ku koncu sú spomenuté vyjadrovacie možnosti a obmedzenia tohoto modelu.

2.1 Popis

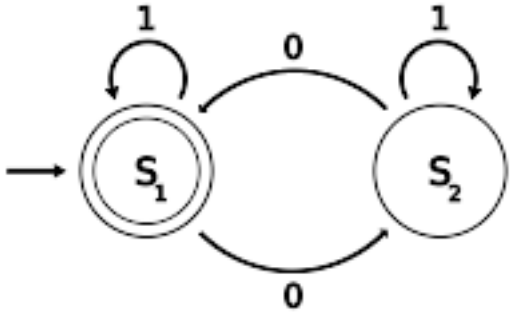
Teória automatov je časť informatiky zaoberajúca sa matematickými modelmi abstraktných automatov a ich použitiu na riešenie rôznych úloh.

Konečný automat je teoretický výpočetný model popisujúci jednoduchý počítač, ktorý môže byť v jednom z konečnej množiny stavov - odtiaľ pochádza aj jeho názov. Medzi stavmi prechádza na základe pravidiel a symbolov, ktoré načítava zo vstupu. Jednoduchý konečný automat si udržiava len informáciu o aktuálnom stave a nemá žiadnu ďalšiu pamäť. Príklad konečného automatu môžeme vidieť na obrázku 2.1.

Každý konečný automat je zložený z piatich súčastí:

- Konečná množina stavov, v ktorých sa automat môže nachádzať
- Abeceda vstupných symbolov, ktoré automat načítava
- Konečná množina pravidiel, ktorými prechádza medzi stavmi
- Začiatočný stav, v ktorom sa automat nachádza pred načítaním symbolu
- Množina koncových stavov, ktorá je podmnožinou všetkých stavov

Automat sa na začiatku nachádza v počiatočnom stave. V každom kroku prečíta jeden symbol zo vstupu, a prejde do stavu, ktorý je daný príslušným pravidlom. Pravidlá sú zložené z aktuálneho stavu, vstupného symbolu a nasledujúceho stavu. Podľa toho, či automat skončí po načítaní reťazca v koncovom stave, hovoríme o tom, či automat reťazec príjma alebo nepríjma. Súbor všetkých príjmaných reťazcov sa nazýva *jazyk*.



Obrázok 2.1: Konečný automat - zápis pomocou grafu.

2.2 Formálna definícia

Aby sme mohli konečný automat formálne popísať, musíme najprv definovať niekoľko základných pojmov.

2.2.1 Abeceda, symboly a reťazce

Abeceda je konečná neprázdna množina prvkov, ktoré sa nazývajú *symboly*. Postupnosť symbolov tvorí *reťazec*. Špeciálnym prípadom je *prázdny reťazec*, označovaný symbolom ϵ, ktorý popisuje reťazec neobsahujúci žiadny symbol. Nasledujúca definícia formálne popisuje reťazec použitím rekurzívnej metódy:

Nech Σ je abeceda. Potom platí:

1. ϵ je reťazec nad Σ.
2. Keď x je reťazec nad Σ, a súčasne symbol $a \in \Sigma$, potom xa je reťazec nad Σ.

2.2.2 Formálny jazyk

Jazyk je množina reťazcov zložených zo symbolov abecedy.

Nech Σ je abeceda a Σ^* popisuje množinu všetkých reťazcov nad abecedou Σ. Potom o podmnožine, pre ktorú platí

$$L \subseteq \Sigma^*$$

hovoríme ako o *jazyku* nad abecedou Σ

2.2.3 Konečný automat

Formálne ho môžeme zapísať ako päticu $M = (Q, \Sigma, R, s, F)$, kde

- Q je konečná množina stavov

- Σ je vstupná abeceda
- $R \subseteq Q(\Sigma \cup \{\epsilon\}) \times Q$ je binárna relácia
- $s \in Q$ je počiatočný stav
- $F \subseteq Q$ je množina koncových stavov

Prvky množiny R sa nazývajú *pravidlá*. Pravidlá $(pa, q) \in R$, kde $p, q \in Q$ a $a \in \Sigma \cup \{\epsilon\}$, sa väčšinou zapisujú vo forme

$$pa \rightarrow q$$

2.3 Použitie

Automaty sú úzko späté s teóriou formálnych jazykov, kde sa používajú na teoretické účely (generovanie a overovanie jazykov), aj na praktické účely (výstavba lexikálnych a syntaktických analyzátorov) Okrem toho však nájdu uplatnenie aj v iných oblastiach informatiky, napríklad:

- Návrh hardware a obvodov
- Sieťové technológie
- Umelá inteligencia
- Multiagentné systémy
- Rozpoznávanie textu
- a iné ...

Táto práca sa však bude zaoberať automatmi takmer výhradne z pohladu formálnych jazykov.

2.4 Spôsob zápisu

2.4.1 Stavový diagram

Exituje niekoľko spôsobov zápisu konečných automatov. Jeden z najčastejších a najviac názorných je pomocou stavového diagramu. Stavový diagram je vlastne orientovaný graf s nasledujúcimi prvkami:

Stavy: sú body grafu väčšinou reprezentované kruhom, v ktorom je napísaný názov príslušného stavu, a poprípade výstupný symbol (u Moorovho automatu).

Vstupné symboly: konečná množina vstupných symbolov, ktoré sú zobrazené nad každou hranou grafu.

Výstupné symboly: umiestnenie výstupných symbolov u prevodníkových automatov (ktoré sú bližšie popísané v kapitole 4.2) je dané ich typom. Zatiaľ čo u Moorovho automatu (viz obrázok 4.3) sa nachádza v stave (čiže stav je popísaný ako A/B, kde A je názov stavu a B je výstupný symbol), zatiaľ čo u Mealyho automatu (viz obrázok 4.4) sa nachádza nad hranou (spolu so vstupným symbolom).

Hrany: reprezentujú pravidlo popisujúce prechod z jedného stavu do druhého podľa určeného vstupného symbolu umiestneného nad hranou. Hrany sú väčšinou zobrazené ako šípky, a smerujú od súčastného stavu do nasledujúceho.

Počiatočný stav: je stav, ktorý je určený šípkou ukazujúcou "odnikiaľ" (názorný príklad na obrázku 2.1).

Konečné stavy: je to množina stavov zobrazená dvojitou kružnicou. Popisuje stavy, v ktorých automat vstupný reťazec príjma.

Stavový diagram je obľúbený hlavne pri výuke, za čo vďačí svojej názornosti.

2.4.2 Tabuľka prechodov

Tabuľka prechodov obsahuje informácie o tom, do akého ďalšieho stavu automat prejde na základe aktuálneho stavu a vstupného symbolu. Tabuľky sú obľúbené hlavne pri návrhu logických obvodov. Poznáme viac typov:

Jednorozmerné tabuľky

Dajú sa pochopiť aj ako pravdivostné tabuľky pre logickú funkciu v boolovej algebre (ako sa aj väčšinou používajú pri návrhu logických obvodov). Vstupný symbol a stavy predstavujú vstup logickej funkcie a ďalší stav spolu s výstupnými symbolmi sa dostane ako výsledok tejto funkcie. Tu je jednoduchý príklad jednorozmernej tabuľky:

A	B	Súčastný stav	Ďalší stav	Výstup
0	0	S_1	S_2	1
0	0	S_2	S_1	0
0	1	S_1	S_2	0
0	1	S_2	S_2	1
1	0	S_1	S_1	1
1	0	S_2	S_1	1
1	1	S_1	S_1	1
1	1	S_2	S_2	0

Dvojrozmerné tabuľky

Dvojrozmerné tabuľky sú asi najčastejšie používané na zápis konečných automatov. Na vertikálnej osi zvyknú byť zobrazené aktuálne stavy, na horizontálnej vstupné symboly, a

medzi nimy nasledujúce stavy. Tu je príklad dvojrozmernej tabulky, ktorá popisuje identický automat, ako v predchádzajúcom príklade:

$\frac{vstup}{stav}$	00	01	10	11
S_1	S_2	S_2	S_1	S_1
S_2	S_1	S_2	S_1	S_2

Ako vidno z príkladu, tento zápis je omnoho efektívnejší čo sa týka veľkosti tabuľky.

2.5 Obmedzenia

Aj keď sú automaty svojou efektivitou a jednoduchosťou jeden z najviac používaných konceptov v informatike, na druhú stranu je však ich vyjadrovacia sila v oblasti formálnych jazykov, ktoré vedia rozpoznať, docela malá - dokážu definovať len regulárne jazyky. Existuje vela jednoduchých jazykov, ktoré konečné automaty nevedia rozpoznať. Klasickým príkladom je jazyk definovaný nasledovne:

$$L = \{a^n b^n : n \geq 1\}$$

Matematický model, ktorý by bol schopný rozpoznať jazyk tohoto typu, by si musel zapamätať počet symbolov a a následne ho porovnať s počtom symbolov b. Konečný automat je síce schopný zapamätať si ľubovolnú informáciu pomocou svojich stavov, ale pretože počet stavov je konečný, tak si automat nemôže zapamätať n, ktoré môže narastať až do nekonečna (reálny príklad - keď nahradíme a symbolom '(' a b symbolom ')', tak dostaneme jednoduchý jazyk popisujúci párovanie zátvoriek v aritmetickom výraze. Konečný automat nieje schopný skontrolovať, či sú v danom výraze zátvorky popárované).

Kapitola 3

História

Jednoduché konečné automaty sa v elektromechanických zariadeniach používali už od minulého storočia. Avšak formálna verzia automatov sa zrodila až vďaka niekoľkým dôležitým vedeckým prácam.

3.1 Turingov stroj

V roku 1936 predstavil Alan Turing svoj matematický model všeobecnáho stroja na spracovanie informácii [8], ktorý dnes poznáme pod pojmom *Turingov stroj*. To bolo niekoľko rokov pred vznikom počítačov takých, aké ich dnes poznáme. Turing vymyslel tento stroj na riešenie zložitých dlhotrvajúcich matematických problémov všeobecne známych pod pojmom *Entscheidungsproblem*, ktorý môžeme voľne preložiť ako *problém rozhodovania*. Konkrétne sa zaoberal oborom predikátovej logiky, kde riešil otázku, či existuje algoritmus, ktorý by dokázal určiť, či je výrok v predikátovej logike teorém alebo nie.

Turingov stroj sa skladal z:

Nekonečnej pásky , na ktorej sú uložené symboly z konečnej abecedy.

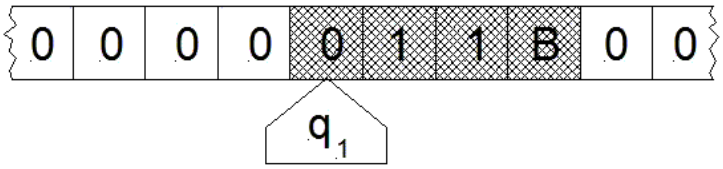
Stavového registru , v ktorom je uložený aktuálny stav.

Čítacej hlavy , ktorá je schopna zapisovať a načítavať aktuálny symbol z pásky.

Tabuľky inštrukcií , kde sú uložené inštrukcie o ďalšom kroku, ktorý je určený podľa vstupného symbolu na páske a aktuálneho stavu uloženého v stavovom registry.

Ako vidíme, srdcom celého stroje je vlastne konečný automat, aj keď vtedy ešte tento pojem nebol zavedený. Schematický náčrt môžeme vidieť na obrázku 3.1.

Turingov stroj nebol „stroj" v pravom slova zmysle, bol to skôr myšlienkový experiment skúmajúci hraničné možnosti mechanického počítania. Stal sa však základným pojmom v teoretickej informantike a študovanie jeho vlastností prináša hlbšie preniknutie do mnohých oblastí v informatike.



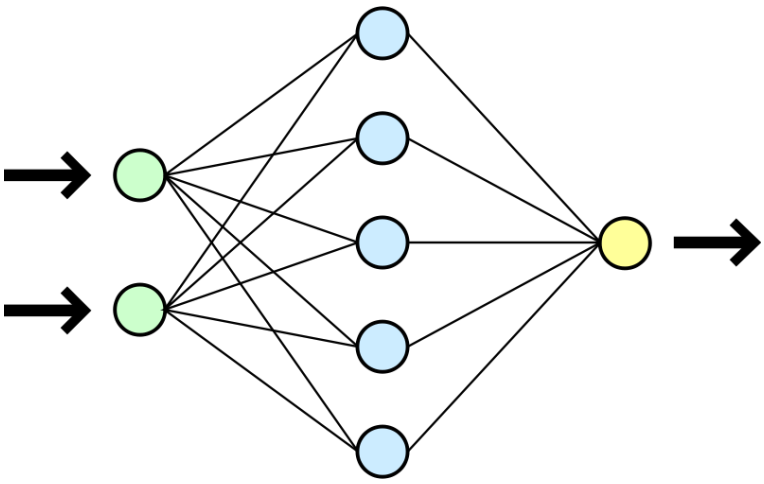
Obrázok 3.1: Turingov stroj - schematický náčrt.

3.2 Neurónové siete

Zatiaľ čo sa Turing venoval vo svojej práci matematicko-logickej stránke, dvaja americký vedci Warren Sturgis McCulloch a Walter Pitts sa nechali inšpirovať vo svojom výskume štruktúrou mozgu. McCulloch, vyštudovaný neurofyziológ, si uvedomil kľúčovú rolu neurónov, ktoré sú základným stavebným kameňom mozgu, pri spracovaní informácií. Spolu s Pittsom vypracovali matematický model neurónu a ukázali ako môže byť použitá sieť týchto neurónov na spracovanie informácií [9]. Hoci sa im ich pokus o simuláciu mozgu na Turingovom stroji nepodaril podľa ich predstáv, podarilo sa im vyvinúť neurónovú sieť (zjednodušenú neurónovú sieť zobrazuje obrázok 3.2), jeden z najužitočnejších konceptov v informatike.

V roku 1951 sa McCullochov a Pittsov model dostal k matematikovi Stephenovi Kleenovi, ktorý na jeho základe prvýkrát popísal jazyk príjmaný ich neurálnou sieťou a vytvoril *Kleenov teorém*, ktorý bol kľúčový pre teóriu automatov [2].

Tieto tri práce dali základ teórii automatov tak ako ju dnes poznáme. Preskúmali automaty ako z praktickej stránky, ktorá viac vyhovuje inžinierom, tak aj z teoretickej, ktorú si obľúbili matematici.



Obrázok 3.2: Zjednodušený príklad neurónovej sieťe

Kapitola 4

Rozdelenie

Automaty môžme rozdeliť do viacerých kategórií podľa rôznych kritérií:

- Deterministické / Nedeterministické
- Rozpoznávacie / Prevodníkové
- Moorove / Mealyho

Každá skupina popisuje inú vlastnosť automatov, čiže sa medzi sebou môžu rôzne kombinovať (napríklad nedeterministický rozpoznávací automat, alebo deterministický prevodníkový Mealyho automat). V nasledujúcej kapitole si postupne popíšeme jednotlivé kritéria a ich výhody a nevýhody.

4.1 Deterministické / Nedeterministické

4.1.1 Nedeterministické konečné automaty

Prechodovú funkciu je možné definovať tak, že v každom pravidle je pre jeden vstupný symbol namiesto jedného stavu celá množina nasledujúcich stavov. Takýto automat sa nazýva *nedeterministický konečný automat*. Takýto hypotetický automat potom pri prečítaní jedného symbolu zo vstupu prejde akoby súčasne do všetkých stavov tejto množiny, a zo všetkých týchto stavov ďalej pokračuje načítavaním ďalších vstupných symbolov. Vstupný reťazec je automatom príjimaný vtedy, ked je aspoň jeden zo stavov, v ktorých sa automat nachádza, prvkom množiny koncových stavov.

Automaty s ϵ-prechodmi

Špeciálny prípad tvoria takzvané *automaty s ϵ-prechodmi*. Tieto obsahujú pravidlá so vstupným symbolom ϵ, ktorý znázorňuje prázdny vstup (ϵ obecne v celej teórii formálnych jazykov označuje prázdne slovo). ϵ-prechody automat prevádza neustále bez načítania vstupného symbolu. Takáto možnosť sa môže zdať ako výhoda oproti automatu bez

ϵ prechodov, avšak výpočetná sila obidvoch automatov je rovnaká. Môžeme povedať, že nedeterministické automaty bez ϵ-prechodov sú vlastne podmnožinou automatov s ϵ-prechodmi.

Formálna definícia

Matematicky môžeme nedeterministický automat s ϵ-pravidlami vyjadriť ako päticu $M = (S, \Sigma, T, s_0, A)$, kde:

- S je konečná množina stavov
- Σ je vstupná abeceda
- T je prechodová funkcia ($T : S \times (\Sigma \cup \{\epsilon\}) \rightarrow P(S)$)
- s_0 je začiatočný stav ($s_0 \in S$)
- A je množina koncových stavov ($A \subseteq S$)

kde $P(S)$ je potenčná množina S, ϵ je prázdny reťazec. Keď X je reťazec nad abecedou Σ, tak M príjma reťazec X vtedy, keď existuje reprezentácia $X = x_1 x_2 \ldots x_n, x_i \in (\Sigma \cup \{\epsilon\})$ a súčasne poradie stavov $r_0, r_1, \ldots, r_n, r_i \in S$ a sú splnené nasledovné podmienky:

1. $r_0 = s_0$
2. $r_i = T(r_{i-1}, x_i)$, pre $i = 1, \ldots, n$
3. $r_n \in A$.

Implementácia

Nedeterministické konečné automaty možno naimplementovať viacerými spôsobmi:

- Prevodom na deterministický konečný automat. Tento prevod je pomerne jednoduchý (príklad viz na obrázkoch 4.1 a 4.2). Ale v najhoršom prípade môže nastať situácia, kde sa počet nových stavov deterministického automatu zvýši exponenciálne. Oproti ostatným riešeniam má ale značnú výhodu v rýchlosti.
- Udržiavaním množiny aktuálnych stavov, v ktorých sa automat práve môže nachádzať. Pri načítanom symbole sa automat pokúsi previesť pre každý jeden stav z množiny pravidlá príslušiace vstupnému symbolu a tomuto stavu. Pri načítaní posledného symbolu potom automat už len skontroluje, či je niektorý stav z množiny aktuálnych stavov aj v množine koncových stavov, a keď taký existuje, tak automat vstupný reťazec príjma. Toto riešenie spotrebuje v najhojršom prípade tolko isto pamäti na uloženie stavov, ako prevedený deterministický automat, ale je jednoduchšie na implementáciu.

- Vytvorením viacerých kópií. V prípade, že sa má automat rozhodnúť medzi n možnými prechodmi, vytvorí $n-1$ kópií samého seba a každá kópia odsimuluje jedno pravidlo a vstúpi do príslušného stavu. Ked sa po načítaní posledného symbolu aspoň jeden z automatov nachádza v konečnom stave, tak automat vstupný reťazec príjma. Ako predchádzajúce postupy, tak aj tento môže v najhoršom prípade zvýšit pamäťové nároky exponenciálne. Avšak jeho výhoda spočíva v paralelných systémoch, kde sa jednotlivé automaty môžu simulovať paralelne.

- Backtrackingom. V situácii, keď si má automat vybrať medzi viacerými prechodmi, odsimuluje tieto prechody po jednom s tým, že keď sa po načítaní vstupného reťazca nachádza v konečnom stave, tak daný reťazec príjma. Keď neexistuje pravidlo, ktoré by vyhovovalo vstupnému symbolu a aktuálnemu stavu, alebo bol načítaný posledný symbol, a automat sa nenachádza v konečnom stave, tak sa automat vráti do predchádzajúceho stavu a načítaný symbol vráti do vstupného reťazca. Táto metóda je užitočná, keď potrebujeme zaistiť informáciu o presnom poradí vykonaných krokov.

4.1.2 Deterministické automaty

Deterministický konečný automat je taký automat, ktorý v množine pravidiel obsahuje len také pravidlá, pre ktoré platí, že pre každý vstupný symbol existuje len jeden nasledujúci stav. Tento fakt je veľmi výhodný pre praktickú implementáciu, ktorá je vďaka nemu menej náročná ako na pamäť, tak na výpočetný čas. Deterministický automat je vlastne len podmnožinou nedeterministického automatu.

Formálna definícia

Matematicky môžeme deterministický automat vyjadriť ako päticu $M = (S, \Sigma, T, s_0, A)$, kde:

- S je konečná množina stavov
- Σ je vstupná abeceda
- T je prechodová funkcia ($T : (S \times \Sigma) \rightarrow S)$)
- s_0 je začiatočný stav ($s_0 \in S$)
- A je množina koncových stavov ($A \subseteq S$)

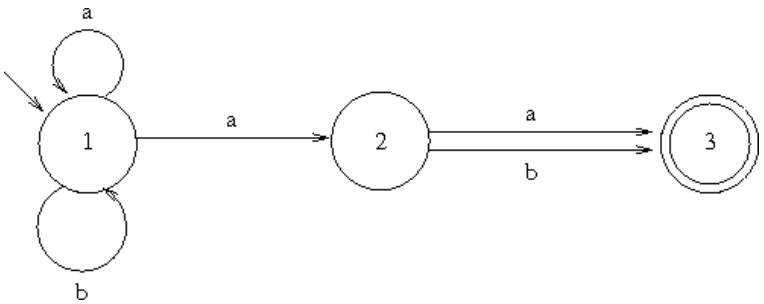
Keď X je reťazec nad abecedou Σ, tak M príjma reťazec X vtedy, keď existuje reprezentácia $X = x_1 x_2 \ldots x_n \in (\Sigma)$ a súčasne poradie stavov $r_0, r_1, \ldots, r_n \in S$ a sú splnené nasledovné podmienky:

1. $r_0 = s_0$
2. $r_{i+1} = T(r_i, x_i)$, pre $i = 1, \ldots, n-1$

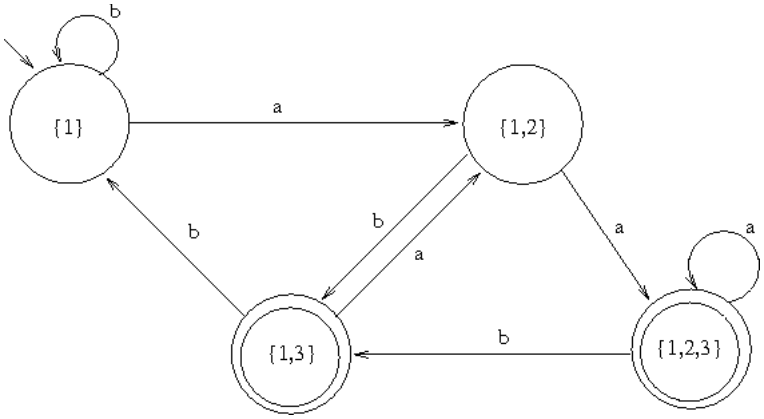
3. $r_n \in A$.

4.1.3 Ekvivalencia

Akokoľvek sa možnosť prevádzať viac výpočtov naraz môže zdať užitočná, v skutočnosti je výpočetný model nedeterministického automatu úplne rovnako mocný ako model deterministického automatu. Obidva príjmajú len regulárne jazyky. Takisto existuje relatívne triviálny mechanický prevod lubovolného nedeterministického automatu na ekvivalentný deterministcký, ale počet stavov môže vzrásť exponenciálne. Každý stav prevedeného deterministického automatu potom odpovedá nejakej množine stavov pôvodného nedeterministického automatu s tým, že sú medzi nimy jednoznačné prechody. Príklad nedeterministického automatu a k nemu ekvivalentného deterministického je na obrázkoch 4.1 a 4.2.



Obrázok 4.1: Príklad nedeterministicého automatu



Obrázok 4.2: Deterministický automat, ktorý vznikol konverziou nedeterministikého automatu z obrázku 4.1

4.1.4 Výhody a nevýhody

Deterministické konečné automaty sú jedným z najpraktickejších modelov v informatike, pretože sa dajú naimplementovat triviálnym algoritmom využívajúc konštantný priestor

a lineárnu zložitost výpočtu. Pracujú nad datovým vstupom znak po znaku bez nutnosti načítania viacerých vstupných symbolov. Práve vďaka týmto vlastnostiam sú deterministické automaty obľúbené hlavne v praktických aplikáciách.

Nedeterministické automaty potrebujú na vyjadrenie toho istého modelu ako deterministické o mnoho menej stavov (pri konverzii na deterministické počet stavov môže narastať exponenciálne). Túto výhodu ocenia hlavne teoretici, ktorý tak majú možnosť jednoducho popísať zložité systémy, avšak ich komplikovaná, a na pamäťový prietor a výpočetný čas náročná implementácia môže byť veľkou nevýhodou pri praktickom využití.

4.2 Rozpoznávacie / Prevodníkové

4.2.1 Rozpoznávacie automaty

Rozpoznávacie automaty (niekedy sa nazývajú aj akceptory) slúžia na príjmanie reťazcov. Ich výsledok je binárny podľa toho, či reťazec príjmajú alebo nie. Pre každý stav vieme povedať, či je konečný, a teda keď v ňom automat skončí, tak reťazec príjma, alebo nieje konečný a automat reťazec nepríjma.

O týchto automatoch môžeme povedať aj to, že rozpoznávajú také jazyky, ktoré obsahujú všetky slová príjmané automatom, a žiadne iné. Taký jazyk sa potom nazýva *regulárny jazyk* a môžeme povedať, že je príjmaný automatom.

Tieto automaty sa používajú hlavne v lexikálnej analýze, kde slúžia vďaka ich vlastnostiam ako nástroj pre rozpoznávanie jednotlivých častí textu alebo zdrojového kódu.

4.2.2 Prevodníkové automaty

Prevodníkové automaty generujú výstup na základe vstupu a daného stavu. Narozdiel od rozpoznávacieho automatu, ktorý má len vstup, majú prevodníkové automaty aj výstup, na ktorý väčšinou vypisujú symboli z výstupnej abecedy. Na tento typ automatov sa môžeme pozrieť aj ako na generátor reťazcov. Potom množinu všetkých reťazcov generovaných automatom môžeme chápať ako jazyk - tým pádom tento automat slúži ako generátor jazykov. Avšak týmto jeho vyjadrovacia sila bohužiaľ nevzrastie, pretože ako príjmaný, tak aj generovaný jazyk patrí do skupiny regulárnych jazykov.

Názov *prevodníkový automat* pochádza z toho, že tento automat vlastne prevádza vstupný reťazec na výstupný pomocou svojich stavov. Pokiaľ je nedeterministický, tak môže nastať situácia, že pre jeden vstupný reťazec sa vygeneruje viacero výstupných reťazcov.

Použitie prevodníkových automatov je veľmi široké, najčastejšie sa s nimi môžeme stretnúť pri návrhu logických obvodov, ale napríklad aj vo výskume spracovania prirodzeného jazyka.

Formálna definícia

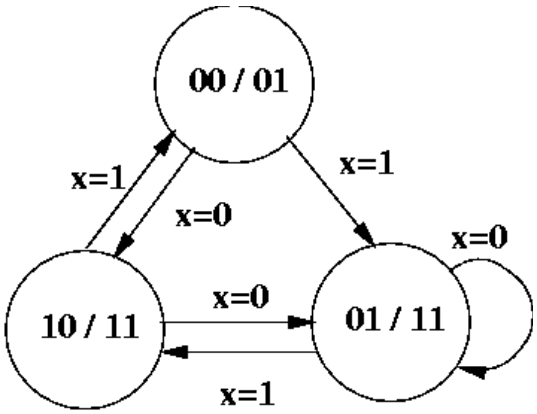
Matematicky môžeme prevodníkové automaty vyjadriť ako šesticu $M = (Q, \Sigma, \Gamma, s_0, F, \delta)$, kde:

- Q je konečná množina stavov
- Σ je vstupná abeceda
- Γ je výstupná abeceda
- s_0 je počiatočný stav ($s_0 \in Q$)
- F je množina koncových stavov ($F \subseteq S$)
- δ je prechodová funkcia ($\delta \subseteq Q(\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times Q$), kde ϵ je prázdny reťazec.

Moorove / Mealyho

Prevodníkové automaty by sme mohli rozdeliť do dvoch základných skupín - Mealyho automaty a Moorove automaty. Tieto dva prístupy sú plne ekvivalentné, a môžeme ich medzi sebou ľubovolne prevádzať. Každý ma svoje výhody aj nevýhody.

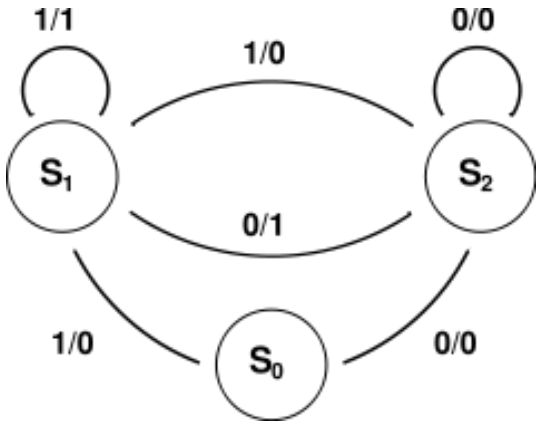
Moorove automaty: Pri Moorových automatoch je výstup daný iba konkrétnym stavom nezávisle od pravidla, ktoré sa použilo ani od vstupného symbolu. Tento model vymyslel priekopník teórie automatov Edward F. Moore [5]. Výhodou Moorových automatov je, že majú jednoduchšie chovanie ako Mealyho, čo je ale vyvážené tým, že na rovnakú funkčnosť potrebujú väčší počet stavov a prechodov medzi nimi. Typický príklad Moorovho automatu môžeme vidieť na obrázku 4.3.



Obrázok 4.3: Moorov automat

Mealyho automaty: Mealyho automat, narozdiel od Moorovho, generuje výstup nielen v závislosti od aktuálneho stavu, ale taktiež na základe vstupného symbolu a použitého

pravidla. Názov Mealyho automatov pochádza od G. H. Mealyho, ktorý tento prístup vymyslel v roku 1955 [3]. Mealyho automaty majú síce komplexnejšie chovanie ako Moorove, ale zato na realizovanie rovnakej funkčnosti potrebujú menší počet stavov. Typický príklad Mealyho automatu môžeme vidieť na obrázku 4.4.



Obrázok 4.4: Mealyho automat

Kapitola 5

Lenivý konečný automat

V tejto kapitole si na úvod neformálne popíšeme, čo je to *lenivý konečný automat*, uvedieme formálnu definíciu a metódu prevodu na obyčajný konečný automat.

V druhej časti bude nasledovať popis implementácie simulačného programu, ktorý sa dá okrem odsimulovania automatu použiť napríklad aj na vystavanie jednoduchého syntaktického analyzátoru.

V poslednej časti sa nachádza úvaha nad možnými rozšíreniami projektu a zhrnutie celej práce.

5.1 Popis

Lenivý konečný automat obsahuje v podstate jednoduché rozšírenie pravidiel. Namiesto jedného symbolu na prechode môže načítať celý reťazec. To je obzvlášť výhodné na zápis pre jednoduchý lexikálny analyzátor, kde uživateľ môže zapisovať jednotlivé reťazce ako prechodové pravidlá. Príklad takéhoto automatu je na obrázku 5.1.

Toto rozšírenie je čiste kozmetického charakteru, lenivý konečný automat má rovnakú vyjadorvaciu silu ako obyčajné konečné autmaty, a ako si ukážeme v ďalšej časti tejto kapitoly, existuje jednoduchý algoritmus na prevod medzi lenivým a obyčajným konečným automatom (5.3).

5.2 Formálna definícia

Lenivý automat môžeme formálne zapísať ako päticu $M = (Q, \Sigma, R, s, F)$, kde:

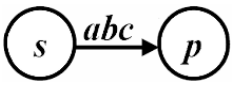
- Q je konečná množina stavov
- Σ je vstupná abeceda
- $R \subseteq Q(\Sigma^* \cup \{\epsilon\}) \times Q$ je množina pravidiel
- $s \in Q$ je počiatočný stav

- $F \subseteq Q$ je množina koncových stavov

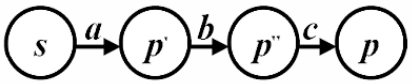
Jediný rozdiel oproti obyčajnému automatu je v množine pravidiel, kde je namiesto vstupného symbolu reťazec Σ^*.

5.3 Prevod

Podstatou prevodu lenivého konečného automatu na automat obyčajný je jednoduchý algoritmus. Každý vstupný reťazec je rozložený na jednotlivé symboly, ktoré sú následne použité na vytvorenie nových pravidiel. Pre každú dvojicu nasledujúcich symbolov je vytvorený nový stav. Cez tieto stavy potom automat prechádza pri príjmaní celého reťazca. Príklad lenivého automatu aj s prevedeným obyčajným automatom môžete vidieť na obrázkoch 5.1 a 5.2.



Obrázok 5.1: Príklad lenivého automatu - zápis pomocou grafu



Obrázok 5.2: Konečný automat získaný prevodom z lenivého automatu z obrázku 5.1

Kapitola 6

Implementácia

V nasledujúcej kapitole bude popísaný návrh, implementácia a použitie aplikácie postavenej na základe lenivého konečného automatu. Jej využitie je aj teoretické, kde je poskytnutá možnosť odsimulovať lenivý konečný automat, na ktorom sa dajú skúmať rôzne vlastnosti lenivých automatov, tak aj praktické, kde môže poslúžiť ako základ syntaktického analyzátoru, keďže na vstupný konfiguračný súbor sa dá pozerať aj ako na gramatiku (jeho formát bol inšpirovaný formátom vstupného súboru programov *GNU Bison* [6] a *yacc*), aj keď s určitými obmedzeniami.

6.1 Použitie

Základom programu je nedeterministický prevodníkový mealyho automat s ϵ-pravidlami. Tento model bol zvolený vzhľadom na jeho možnosť jednoduchého zápisu uživateľom, ktorá sa hodí na dané simulácie. Na každej hrane si uživateľ môže zadať vlastný kód v jazyku C, ktorý sa spustí vždy, keď automat prejde danou hranou. Vstupom pre automat je konfiguračný súbor, ktorý je bližšie popísaný v časti 6.1.1. Tento súbor obsahuje popis automatu. Konfiguracný súbor je načítaný parserom, ktorý na základe neho vygeneruje konečný automat. Na základe tohoto automatu sa potom vygeneruje výstupný súbor v jazyku C, ktorý po preklade daný automat odsimuluje. Samotný konfiguračný súbor je jediným vstupom pre program. Ten má dva povinné argumenty, a to názov konfiguračného súboru a výstupného súboru, do ktorého sa vygeneruje zdrojový kód výsledného automatu.

6.1.1 Konfiguračný súbor

Ako už bolo spomenuté, konfiguračný súbor má formát veľmi podobný zdrojovému súboru pre program *GNU Bison*. Skladá sa z jednotlivých príkazov začínajúcich názvom aktuálneho stavu, a ukončených znakom ';'. Príkazy obsahujú jednotlivé pravidlá a sú nasledujúceho formátu:

```
aktualny_stav : ‘‘vstupny_retazec’’ dalsi_stav {uzivatelov_kod}
```

```
| ‘‘vstupny retazec’’ dalsi_stav {uzivatelov_kod}
| ...
;
```

Tento riadok popisuje dve hrany, ktoré prechádzajú z jedného stavu do druhého načítaním reťazca uzavreného v úvodzovkách. Pri každom prechode touto hranou zároveň spustí uživateľom definovaný kód napísaný v zložených zátvorkách. Nasledujúci zoznam obsahuje krátky popis ku každej zložke príkazu:

- `aktualny_stav` : Názov aktuálneho stavu, pre ktorý sú dané pravidlá definované. Názov môže obsahovať len znak, číslicu alebo podtržítko. Táto informácia je *povinná* na začiatku každého príkazu.
- `':'` : Dvojbodka oddeluje názov aktuálneho stavu od zvyšku príkazu popisujúceho jednotlivé pravidlá. Táto informácia je *povinná*.
- "`vstupny_retazec` ": Musí byť uzavretý v dvojitých úvodzovkách. Môže obsahovať ľubovolné znaky, a zároveň podporuje nasledujúce špeciálne znaky, ktoré prekladá na príslušné hodnoty:
  - `'\n'` : koniec riadku
  - `'\t'` : horizontálny tabulátor
  - `'\v'` : vertikálny tabulátor
  - `'\"'` : týmto znakom sa dajú definovať dvojité úvodzovky, ktoré normálne ohraničujú vstupný reťazec.
  - `'\\'` : kombinácia dvoch opačných lomítok popisuje obyčajné opačné lomítko, ktoré je normálne používané na odlíšenie špeciálnych znakov.

  Definovanie vstupného reťazca je *nepovinné*, a keď je vynechané, tak sa vstupný reťazec považuje za prázdny.
- `dalsi_stav` : Názov ďalšieho stavu, do ktorého automat prejde po načítaní reťazca. Táto informácia je *nepovinná*, ale keď je vynechaná, značí to, že po načítaní vstupného reťazca sa automat dostáva do konečného stavu.
- `{uzivatelov_kod}` : Medzi zloženými zátvorkami je uvedený uživateľov kód v jazyku C, ktorý sa vykoná vždy po odsimulovaní pravidla. Táto informácia je *nepovinná*, a keď je vynechaná, tak sa žiadny kód neprevedie.
- `'|'` : Tento znak oddeluje ďalšie pravidlá od predchádzajúceho. Pre každé ďalšie pravidlo platí, že má rovnakú syntax od symbolu ':'. Táto informácia je *nepovinná*, príkaz môže mať len jedno pravidlo, a ďalšie pravidlá nemusia byť definované.
- `';'` : Symbol ukončenia príkazu je *povinný* na konci každého príkazu.

V konfiguračnom súbore sa môže na ľubovolnom mieste nachádzať komentár, ktorý má syntax identickú ako v jazyku C - t.j. všetko uzavreté v kombinácii znakov `/*` a `*/`. Príklad jednoduchého konfiguračného súboru, ktorý načítava príkaz `mov` v jazyku assembler v dvoch variáciách (`mov ax, bx` a `mov bx, ax`) a na standardný výstup píše binárne hodnoty podľa vstupu:

```
start   : ‘‘ ’’          start
        | ‘‘\n’’         start
        | ‘‘mov’’        mov     { printf(‘‘000’’); }
        | ‘‘’’                   { printf(‘‘\n’’); }        /* finish */
        ;

mov     : ‘‘ ’’          mov
        | ‘‘ax, ’’       movax   { printf(‘‘00’’); }
        | ‘‘bx, ’’       movbx   { printf(‘‘01’’); }
        ;

movax   : ‘‘ ’’          movax
        | ‘‘bx’’         endl    { printf(‘‘01’’); }
        ;

movbx   : ‘‘ ’’          movbx
        | ‘‘ax’’         endl    { printf(‘‘00’’); }
        ;

endl    : ‘‘\n’’         start
        | ‘‘’’                   { printf(‘‘\n’’); }       /* finish */
        ;
```

6.2 Návrh

Návrh programu využíva niektoré rysy objektovo-orientovaného prístupu, ktoré sú kombinované s imperatívnym prístupom. Skladá sa z troch hlavných modulov:

- Parser konfiguračného súboru 6.2.1
- Konečný automat 6.2.2
- Generátor výstupného kódu 6.2.3

Ich vzájomné prepojenie je znázornené na obrázku 6.1. Zatiaľ čo konečný automat je definovaný ako trieda, ktorá má metódy na pridávanie stavov, pravidiel, ich vypisovanie

alebo uvolňovanie z pamati, parser a generátor sú naimplementované ako procedúry, ktoré pracujú nad týmto automatom.

Pre potreby týchto modulov boli naprogramované pomocné funkcie a triedy, ktoré sú využívané po celom zdrojovom kóde. Medzi ne patrí:

error Tento modul obsahuje pomocné funkcie na vypisovanie chybových hlášok, ale aj na alokáciu pamäti alebo kopírovanie stringu do nového miesta v pamäťi. Tieto funkcie ale pri neúspechu končia termináciou programu a vypísaním chybových hlášok (tento prístup výrazne zmenšuje dĺžku napísaného kódu a súčasne nutnosť kontroly úspešného alokovania pamati, čím zjednodušuje písanie a ladenie celého projektu), preto sú obsiahnuté práve v tomto module.

list Tento modul obsahuje jednoduchú triedu určenú na prácu s lineárnym zoznamom. Táto trieda je použitá hlavne konečným automatom pre uloženie pravidiel a stavov. Je však navrhnutá všeobecne tak, aby jednotlivé položky mohli obsahovať ľubovolné údaje.

start.c Tento súbor nieje modul, ale vkladá sa do výsledného kódu vygenerovaného programom. Obsahuje jednoduchý zásobník, do ktorého sa ukladá poradie použitých pravidiel simulovaného automatu. Do súboru je možné pridávať rôzne hlavičky knižníc alebo uživateľské funkcie, ktoré môžu byť potom použité simulovaným automatom.

Nasledujúca sekcia obsahuje krátky popis jednotlivých modulov:

6.2.1 Parser konfiguračného súboru

Tento modul je zodpovedný za správne načítanie konfiguračného súboru. Jeho rozhranie tvorí jediná funkcia `parse()`, ktorá má dva argumenty - konfiguracný súbor a konečný automat. Táto funkcia po zavolaní naplní konečný automat. Skladá sa z lexikálneho analyzátoru, ktorý načítava jednotlivé slová oddelenné medzerami, ktoré predáva syntaktickému analyzátoru. Ten potom kontroluje správnosť syntaxe a zároveň ukladá informácie na pridanie stavu. Akonáhle narazí na znak `'|'` alebo `';'`, zavolá funkciu konečného automatu určenú na pridávanie pravidiel, a predá jej načítané hodnoty. Pri chybnom formáte konfiguračného súboru vypíše chybovú hlášku a program sa ukončí.

6.2.2 Konečný automat

Tento modul obsahuje triedu konečného automatu, ktorá obsahuje tri triedy potrebné na uloženie všetkých informácií vrátane stavov, prechodov medzi nimy a uživateľským kódom. Automat sa tvári navonok ako lenivý, ale vo vnútri má architektúru obyčajného automatu. Rozšírené pravidlá obsahujúce reťazce sú automaticky konvertované na jednoduché pravidlá obsahujúce jeden symbol. Nasledujúci zoznam popisuje triedy použité na uloženie všetkých informácii o automate:

t_rule v tejto triede sú uložené informácie o jednom prechode - *vstupný symbol*, *nasledujúci stav*, *identifikačné číslo prechodu* použité pri ukladaní prechodu na zásobník v samotnej simulácii, a reťazec obsahujúci *uživateľov kód*.

t_state obsahuje *názov stavu*, ktorý sa kontroluje pri parsovaní konfiguračného súboru (aby sa jeden stav nevložil viac krát), *identifikačné číslo*, ktoré slúži na vnútornú reprezentáciu stavu, a nakoniec *zoznam prechodov* obsahujúci objekty triedy `t_rule`.

t_fsm je základná trieda použitá pri práci s automatom. Táto trieda obsahuje celé rozhranie k automatu. Skladá sa zo zoznamu stavov a pomocných údajov pre vytváranie nových stavov (použitých pri konverzii z rozšírených pravidiel na jednoduché) a pre pridelovanie identifikačných čísiel jednotlivým uživateľským kódom.

Samotné rozhranie automatu obsahuje päť verejných metód - konštruktor pre vytvorenie samotných objektov, metódy pre vyhladávanie existujúceho stavu, pridanie nového stavu, pridanie nového pravidla, a deštruktor pre uvolnenie pamäti v ktorej bol objekt použitý.

6.2.3 Generátor výstupného kódu

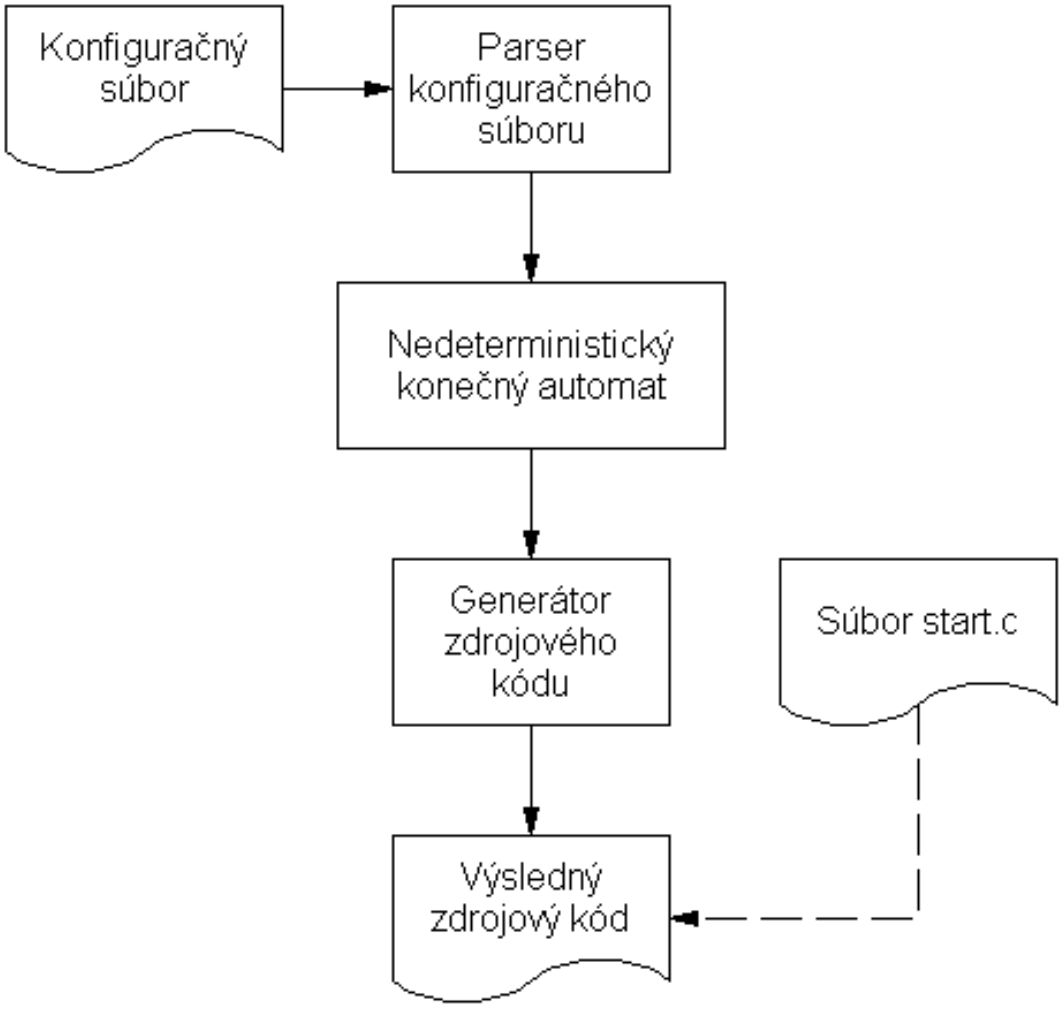
Tento modul obsahuje všetko potrebné pre vygenerovanie zdrojového kódu na základe automatu. Jeho rozhranie tvorí jedna funkcia, ktorá má dva parametre - výstupný súbor a konečný automat. Funkcia po zavolaní vytvorí výstupný zdrojový kód používaný na simuláciu samotného automatu (výstupný súbor je popísaný na 6.2.4).

6.2.4 Výstupný súbor

Pre simuláciu výsledného automatu bola zvolená metóda backtrackingu (bližšie popísaná v odstavci 4.1.1). Vzhľadom na prevodníkový charakter automatu, kde má každá hrana sémantiku v podobe uživateľom definovaného zdrojového kódu, by bol prevod na deterministický automat veľmi náročný, keďže daný algoritmus nieje známy. Avšak backtracking poskytuje požadovanú funkčnosť a súčasne jednoduchšiu implementáciu za cenu nižšej rýchlosti.

Automat funguje na princípe prehladávania stavového priestoru do hĺbky. Snaží sa nájsť úspešnú cestu ku konečnému stavu. Pre prvú takúto nájdenú cestu potom odsimuluje jednotlivé prechody spolu s uživateľským kódom.

Každý stav automatu je reprezentovaný funkciou, v ktorej sa načíta vstupný symbol, a postupne sa simulujú jednotlivé pravidlá volaním funkcií náležitých stavov. Pre každý prechod sa uloží do zásobníka identifikačné číslo použitého pravidla. Tento zásobník je potom použitý na volanie príslušného uživateľského kódu. Keď program narazí na znak konca súboru a zároveň sa nachádza v konečnom stave, tak odsimuluje pravidlá v zásobníku a úspešne ukončí program. V prípade, že narazí na symbol, pre ktorý neexistuje pravidlo v aktuálnom stave, tak prejde do ϵ-pravidiel. Keď takéto pravidlá neexistujú pre daný stav, potom vráti načítaný vstupný symbol do vstupného reťazca a vráti sa do predchádzajúceho



Obrázok 6.1: Návrh aplikácie

stavu použitím návratovej funkcie `return`. Tam potom prechádza ďalšie pravidlá. Program končí neúspechom pri návrate do funkcie `main`, čo znamená, že nebola nájdená žiadna cesta do konečného stavu pre daný vstupný reťazec.

Použitie ϵ-pravidiel je nebezpečné, pretože automat sa môže zacykliť pri určitých konfiguráciách a vstupných reťazcoch. Toto je však ošetrené v programe metódou počítania po sebe nasledujúcich použitých ϵ-pravidiel. Keď tento počet prevýši počet existujúcich stavov, simulácia neskúma generovaný stavový priestor ďalej do hĺbky, ale prechádza na ďalší stav.

Kapitola 7

Záver

Výsledkom tejto bakalárskej práce je aplikácia simulujúca model lenivého konečného automatu. Tento program zároveň slúži aj ako generátor syntaktických analyzátorov v jazyku C s obmedzenou gramatikou.

Aktuálna verzia programu spĺňa základnú funkčnosť, a zároveň obsahuje aj dodatočné vylepšenia ako napríklad detekciu zacyklenia. Ako možné ďalšie rozšírenia by mohol obsahovať možnosť prevodu nedeterministického automatu na plne deterministický, aj keď s určitými obmedzeniami pre zadávané konfigurácie.

Takisto uživateľské rozhranie obsahuje len základnú funkčnosť. Medzi jeho možné rozšírenia by sme mohli zaradiť podrobnejšie hlášky o chybách v konfiguračnom súbore, alebo možnosť definovania skupiny vstupných reťazcov na prechod do ďalšieho stavu (príklad z praxe: definovanie skupiny *bielych znakov* t.j. medzerník, koniec riadku, tabulátor, atd.).

Nieje v možnostiach tejto práce obsiahnuť všetky informácie na danú tému. O automatoch sa môžete dozvedieť viac napríklad v knihe od Alexandra Meduny [4] alebo Jamesa Andersona [1] spomenutej na konci tejto práce. Ako odrazový mostík pre začiatočníkov môže poslúžiť aj Wikipedia [7], kde sa nachádza veľa užitočných, aj keď nie tak podrobných a kvalitných informácií.

Literatura

[1] James A. Anderson. *Automata Theory with Modern Application.* Cambridge University Press, New York, 2006.

[2] Stephen C. Kleene. *Representation of events in nerve nets and finite automata, in Automata studies.* Princeton University Press, 1956.

[3] G. H. Mealy. *A Method for Synthesizing Sequential Circuits.* Bell System Tech. J., 1955.

[4] Alexander Meduna. *Automata and Languages: Theory and Applications.* Springer, London, 2000.

[5] Edward F. Moore. *Gedanken-experiments on Sequential Machines.* Princeton University Press, 1956. Automata Studies.

[6] WWW stránky. Gnu bison manual. `http://www.gnu.org/software/bison/manual/html_mono/bison.html`.

[7] WWW stránky. Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Automata_theory`.

[8] Alan M. Turing. *On computable numbers with an application to the Entscheidungsproblem.* Proceedings of the London Mathematical Society 2, 1936.

[9] Walter Pitts Warren S. McCulloch. *A logical calculus of the ideas immanent in nervous activity.* Bulletin of Mathematical Biophysics 5, 1943.

Zoznam príloh

A Datový nosič CD s kompletnou implementáciou programu, uživateľskou príručkou a zdrojovými kódmi tejto práce v elektronickej podobe.