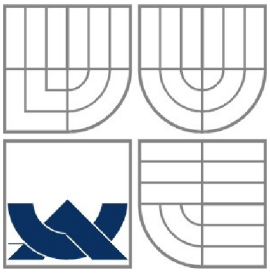
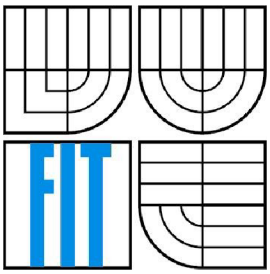


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA SPECIÁLNÍCH MODELECH

SYNTACTIC ANALYSIS BASED ON SPECIAL MODELS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

EVA ZÁMEČNÍKOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

ING. ROMAN LUKÁŠ, PH.D.

BRNO 2007

Zadání bakalářské práce/5729/2006/xzamec10

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Zámečníková Eva**
Obor: Informační technologie
Téma: **Syntaktická analýza založená na speciálních modelech**
Kategorie: Teorie informatiky

Pokyny:

1. Seznamte se podrobně s metodami syntaktické analýzy a s formálními modely, na jejichž bázi pracují.
2. Navrhněte vlastní metodu syntaktické analýzy založené na speciálních fomálních modelech jako např. na sytémech konečných automatů či jiných modelech.
3. Studujte praktické využití syntaktické analýzy založené na modelech z předchozího bodu.
4. Danou metodu implementujte.
5. 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

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Slečna**

Jméno a příjmení: **Eva Zámečníková**
Id studenta: 84093
Bytem: Chaloupky 204, 768 61 Bystřice pod Hostýnem
Narozena: 26. 06. 1985, Kroměříž
(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: Syntaktická analýza založená na speciálních modelech
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

Bakalářká práce se zabývá syntaktickou analýzou pomocí párových automatů. Konečný párový automat obsahuje vstupní a výstupní automat. Vstupní automat provádí syntaktickou analýzu se vstupním řetězcem. Použitá pravidla vstupního automatu řídí výstupní automat, který generuje výstupní řetězec. Hlavním tématem této práce je determinizace vstupního automatu bez ztráty informací o pravidlech použitých v původním automatu.

Klíčová slova

konečný automat, líný konečný automat, konečný převodník, líný konečný převodník, párový konečný automat, nejednoznačnost, determinismus, C++, BISON--.

Abstract

This bachelor thesis is dealing with translation based on coupled finite automaton. Coupled finite automaton contains input and output automaton. The input automaton makes syntactic analysis with an input string. Used rules from the input automaton control the output automaton, which generates an output string. Basic topic of this thesis is determinisation of the input automaton without loss of information about rules used in original automaton.

Keywords

finite automaton, lazy finite automaton, finite transducer, lazy finite transducer, coupled finite automaton, ambiguity, determinism, C++, BISON--.

Citace

Eva Zámečníková: Syntaktická analýza založená na speciálních modelech, bakalářská práce, Brno, FIT VUT v Brně, 2007

Syntaktická analýza založená na speciálních modelech

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením Ing. Romana Lukáše, Ph.D.
Další informace mi poskytli Ing. Karel Masařík, prof. Ing. Tomáš Hruška, CSc. a doc. Dr. Ing. Dušan Kolář. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala. Bakalářská práce byla vypracována na základě veřejně přístupného kódu programu TinyXml a Eclipse.

……
Eva Zámečníková
29.4.2007

Poděkování

Na tomto místě bych chtěla poděkovat Ing. Romanu Lukášovi, Ph.D. za odbornou pomoc při vytváření této práce a za ochotu a čas, který mi věnoval.

Obsah

1 Úvod

Hlavním úkolem teoretické informatiky je popsat pomocí různých druhů formálních modelů tzv. formální jazyky a jejich překlady. V teoretické informatice byly zavedeny různé typy modelů. Tyto modely mohou být různě složité a ve většině případů platí, že čím složitější model použijeme, tím rozsáhlejší třídu jazyků, respektive složitější překlady popíšeme. Problém ale nastává z hlediska implementace. Tam zase naopak platí, že čím jednodušší model použijeme, tím jednodušeji můžeme tento model naimplementovat a tím bude překlad probíhat rychleji. Proto má smysl se zamyslet, zda není možné nějakým způsobem využít pouze jednoduchých modelů, které se snadno implementují, avšak jistým „vylepšením“ přesto zvýšit jejich sílu. Přirozeně nás napadne myšlenka, že můžeme tento jeden jednoduchý model nahradit celým systémem těchto jednoduchých modelů, které jakýmsi způsobem budou spolu komunikovat a tedy vzájemně jeden bude řídit činnost jiného.

Tato bakalářská práce se bude především zabývat systémem modelů líných konečných automatů, který bude konkrétně obsahovat dva prvky, a pomocí tohoto systému bude prováděn překlad. Tento model je nazván párovým konečným automatem. Jeden z těchto automatů je nazýván vstupním automatem, druhý výstupním. Překlad pomocí párového automatu probíhá potom následovně: Vstupní automat postupně načítá vstupní řetězec ze vstupu. Toto načítání jistým způsobem řídí činnost výstupního automatu, který naopak bude generovat výstupní řetězec na výstup. Dá se ukázat, že tento překlad je obecnější, než kdyby byl provedený například jen konečným převodníkem. Tento překlad může být tedy rozdělen do dvou nezávislých fází:

1. fáze – provedení syntaktické analýzy vstupního řetězce vstupním automatem a podle toho, která pravidla byla použita, nastavení řízení výstupního automatu.
2. fáze – nagenerování výstupního řetězce výstupním automatem.

Daleko složitější je tedy 1. fáze, které bude věnována převážná část této bakalářské práce. Tuto fázi je totiž z důvodu požadavku na rychlost potřeba udělat deterministickým způsobem. Je pravda, že existuje spousta algoritmů, které transformují daný konečný automat na deterministický. V našem případě jsou ale nepoužitelné, neboť tato transformace může kompletně změnit strukturu daného automatu a tím je tento nový automat nepoužitelný pro řízení výstupního automatu. Z toho důvodu vznikly v rámci této bakalářské práce nové algoritmy, které tento problém budou řešit.

Struktura bakalářské práce je následující:

- První kapitola obsahuje samotný úvod. Tato část čtenáře neformálně uvede do problematiky párových automatů.
- Ve druhé kapitole je čtenář seznámen se základními pojmy a definicemi z teorie formálních jazyků, které jsou dále používány v ostatních kapitolách.
- Ve třetí kapitole je popsána metoda, která převádí líné konečné převodníky na ekvivalentní deterministické převodníky. Tohoto je využito právě pro vytvoření syntaktického analyzátoru pro daný konečný automat.
- Ve čtvrté kapitole je navržen jazyk a překladač BISON--, který simuluje překlad konečného převodníku.
- V páté kapitole je uvedena rekapitulace výsledků této bakalářské práce

2 Základní pojmy

V této kapitole jsou uvedeny základní pojmy, které jsou potřebné pro pochopení dalšího textu. Jedná se o základní pojmy teoretické informatiky.

2.1 Definice základních pojmů

2.1.1 Definice abecedy

Abeceda je neprázdná konečná množina prvků, které nazýváme symboly.

2.1.2 Definice řetězce nad danou abecedou

Nechť Σ je abeceda. Potom:

- ε je řetězec nad abecedou Σ.
- Jestliže x je řetězec nad abecedou Σ, $a \in \Sigma$, potom xa je řetězec nad abecedou Σ.

Poznámky:

1) Symbol ε značí *prázdný řetězec*. Prázdný řetězec je takový řetězec, který neobsahuje žádný symbol.

2) Symbolem Σ^* budeme značit množinu všech řetězců nad abecedou Σ.

2.1.3 Definice délky řetězce

Nechť x je řetězec nad abecedou Σ. Délka řetězce x, $|x|$, je definována následovně:

- Pokud $x = \varepsilon$, potom $|x| = 0$.
- Pokud $x = a_1 a_2 \ldots a_n$, kde $a_i \in \Sigma$ pro všechna $i = 1, \ldots, n$, potom $|x| = n$.

2.1.4 Definice binární operace konkatenace

Nechť x, y jsou dva řetězce nad abecedou Σ. Konkatenací řetězce x s řetězcem y vznikne řetězec xy (připojením řetězce y za řetězec x). Operace konkatenace je asociativní, ale není komutativní.

2.1.5 Definice reverzace řetězce

Nechť x je řetězec nad abecedou Σ. Reverzace řetězce x, *reverse*(x), je definována následovně:

- Pokud $x = \varepsilon$, potom *reverse*(x) = ε.
- Pokud $x = a_1 a_2 \ldots a_n$, kde $a_i \in \Sigma$ pro všechna $i = 1, \ldots, n$, potom *reverse*(x) = $a_n \ldots a_2 a_1$.

2.1.6 Definice prefixu řetězce

Nechť x, y jsou dva řetězce nad abecedou Σ. x nazveme *prefixem* řetězce y, pokud existuje řetězec z nad abecedou Σ, pro který platí: $xz = y$.

2.1.7 Definice sufixu řetězce

Nechť x, y jsou dva řetězce nad abecedou Σ. x nazveme *sufixem* řetězce y, pokud existuje řetězec z nad abecedou Σ, pro který platí: $zx = y$.

2.1.8 Definice podřetězce

Nechť x, y jsou dva řetězce nad abecedou Σ. x nazveme *podřetězcem* řetězce y, pokud existují řetězce z, z' nad abecedou Σ, pro které platí: $zxz' = y$.

2.1.9 Definice formálního jazyka

Nechť je dána abeceda Σ. Potom množinu L, pro kterou platí $L \subseteq \Sigma^*$, nazveme *formálním jazykem* nad abecedou Σ.

2.1.10 Definice překladu

Nechť Σ, Ω jsou abecedy. Σ budeme nazývat tzv. *vstupní abecedou*, Ω budeme nazývat tzv. *výstupní abecedou*. *Překladem* jazyka $L_{in} \subseteq \Sigma^*$ do jazyka $L_{out} \subseteq \Omega^*$ nazveme libovolnou relaci τ z L_{in} do L_{out}. Jazyk L_{in} nazveme jazykem vstupním, jazyk L_{out} nazveme jazykem výstupním. Pokud pro řetězce $x \in L_{in}$ a $y \in L_{out}$ platí $y \in \tau(x)$, pak řekneme, že řetězec y je výstupem pro řetězec x.

2.2 Definice vybraných formálních modelů

V této kapitole jsou nadefinovány některé základní formální modely, pomocí kterých budeme specifikovat formálních jazyky a překlady. Tyto modely budou využity k překladu v následujících kapitolách.

2.2.1 Pravá lineární gramatika

2.2.1.1 Definice pravé lineární gramatiky

Pravá lineární gramatika je čtveřice $G = (N, T, P, S)$, kde

- N je konečná množina nonterminálních symbolů,
- T je konečná množina terminálních symbolů, přičemž $N \cap T = \emptyset$,
- P je konečná množina pravidel tvaru $A \to xB$, kde $A, B \in N, x \in T^*$ nebo

$$A \to x, \text{ kde } A \in N, x \in T^*,$$

- S je počáteční nonterminální symbol.

2.2.1.2 Definice přímé derivace u pravé linerární gramatiky

Nechť $G = (N, T, P, S)$ je pravá lineární gramatika, nechť $u, v \in (N \cup T)^*$ a $p: A \to x \in P$ je pravidlo. Pak říkáme, že *uAv přímo derivuje uxv* podle pravidla p a zapisujeme $uAv \Rightarrow uxv$ $[p]$ nebo také zkráceně $uAv \Rightarrow uxv$.

2.2.1.3 Definice sekvence derivací u pravé lineární gramatiky

Nechť $G = (N, T, P, S)$ je pravá lineární gramatika.

- Nechť $u \in (N \cup T)^*$. Pak říkáme, že *u derivuje v 0-krocích u* a zapisujeme $u \Rightarrow^0 u$ $[\varepsilon]$ nebo také zkráceně $u \Rightarrow^0 u$.
- Nechť $u_0, u_1, \ldots, u_n \in (N \cup T)^*$, nechť pro všechna $i = 1, ..., n$ platí: $u_{i-1} \Rightarrow u_i$ $[p_i]$. Pak říkáme, že u_0 *derivuje v n-krocích* u_n a zapisujeme $u_0 \Rightarrow^n u_n$ $[p_1 p_2 \ldots p_n]$ nebo také zkráceně $u_0 \Rightarrow^n u_n$.
- Nechť $u \Rightarrow^n v$ $[\pi]$ pro nějaké $n \geq 1$; $u, v \in (N \cup T)^*$. Pak říkáme, že *u netriviálně derivuje v* a zapisujeme $u \Rightarrow^+ v$ $[\pi]$ nebo také zkráceně $u \Rightarrow^+ v$.
- Nechť $u \Rightarrow^n v$ $[\pi]$ pro nějaké $n \geq 0$; $u, v \in (N \cup T)^*$. Pak říkáme, že *u derivuje v* a zapisujeme $u \Rightarrow^* v$ $[\pi]$ nebo také zkráceně $u \Rightarrow^* v$.

2.2.1.4 Definice jazyka generovaného pravou lineární gramatikou

Nechť $G = (N, T, P, S)$ je pravá lineární gramatika. Jazyk generovaný pravou lineární gramatikou G (budeme jej označovat $L(G)$) je definován následovně:

$$L(G) = \{w : w \in T^*, S \Rightarrow^* w\}.$$

2.2.2 Líný konečný automat

2.2.2.1 Definice líného konečného automatu

Líný konečný automat je pětice $M = (Q, \Sigma, R, s, F)$, kde

- Q je konečná množina stavů,
- Σ je vstupní abeceda,
- R je konečná množina pravidel tvaru $px \rightarrow q$, kde $p, q \in Q, x \in \Sigma^*$,
- $s \in Q$ je počáteční stav,
- $F \subseteq Q$ je množina koncových stavů.

2.2.2.2 Definice konfigurace líného konečného automatu

Nechť $M = (Q, \Sigma, R, s, F)$ je líný konečný automat. Pak *konfigurací* nazveme libovolný řetězec $\chi = px$, kde $p \in Q, x \in \Sigma^*$.

2.2.2.3 Definice přímého přechodu

Nechť $M = (Q, \Sigma, R, s, F)$ je líný konečný automat, nechť $\chi = pxu$, $\chi' = qu$ jsou konfigurace, kde $p, q \in Q, u, x \in \Sigma^*$. Nechť $r: px \rightarrow q \in R$ je pravidlo. Pak říkáme, že M provede *přímý přechod* z konfigurace χ do χ' podle pravidla $r: px \rightarrow q \in R$ a zapisujeme $\chi \Rightarrow \chi' [r]$ nebo zkráceně $\chi \Rightarrow \chi'$.

2.2.2.4 Definice sekvence přímých přechodů

Nechť $M = (Q, \Sigma, R, s, F)$ je líný konečný automat.

- Nechť χ je konfigurace, pak M provede *0-přechodů* z χ do χ, zapisujeme $\chi \Rightarrow^0 [\varepsilon]$ nebo také zkráceně $\chi \Rightarrow^0 \chi$.
- Nechť $\chi_0, \chi_1, \ldots, \chi_n$ jsou konfigurace, kde $n \geq 1$ a $\chi_{i-1} \Rightarrow \chi_i [r_i]$, $r_i \in R$ pro všechna $i = 1, \ldots, n$. Pak říkáme, že M provede *n-přechodů* z χ_0 do χ_n a zapisujeme $\chi_0 \Rightarrow^n \chi_n [r_1 \ldots r_n]$ nebo také zkráceně $\chi_0 \Rightarrow^n \chi_n$.
- Nechť $\chi_0 \Rightarrow^n \chi_n [\rho]$ pro nějaké $n \geq 1$, pak říkáme, že χ_0 *netriviálně derivuje* χ_n a zapisujeme $\chi_0 \Rightarrow^+ \chi_n [\rho]$ nebo také zkráceně $\chi_0 \Rightarrow^+ \chi_n$.
- Nechť $\chi_0 \Rightarrow^n \chi_n [\rho]$ pro nějaké $n \geq 0$, pak říkáme, že χ_0 *derivuje* χ_n a zapisujeme $\chi_0 \Rightarrow^* \chi_n [\rho]$ nebo také zkráceně $\chi_0 \Rightarrow^* \chi_n$.

2.2.2.5 Definice jazyka přijímaného líným konečným automatem

Nechť $M = (Q, \Sigma, R, s, F)$ je líný konečný automat. Jazyk přijímaný líným konečným automatem M (budeme jej označovat $L(M)$) je definován následovně:

$$L(M) = \{x : sx \Rightarrow^* f, x \in \Sigma^*, f \in F\}.$$

2.2.2.6 Definice jednoznačného líného konečného automatu

Nechť $M = (Q, \Sigma, R, s, F)$ je líný konečný automat. Pak říkáme, že líný konečný automat M je *jednoznačný*, pokud pro každé $x \in L(M)$ existuje právě jedna posloupnost pravidel ρ taková, že $sx \Rightarrow^* f\,[\rho], f \in F$.

2.2.3 Líný konečný převodník

2.2.3.1 Definice líného konečného převodníku

Líný konečný převodník je šestice $M = (Q, \Sigma, \Omega, R, s, F)$, kde:

- Q je konečná množina stavů,
- Σ je vstupní abeceda,
- Ω je výstupní abeceda,
- R je konečná množina pravidel tvaru $px \rightarrow yq$, kde $p, q \in Q, x \in \Sigma^*, y \in \Omega^*$,
- $s \in Q$ je počáteční stav,
- $F \subseteq Q$ je množina koncových stavů.

2.2.3.2 Definice konfigurace líného konečného převodníku

Nechť $M = (Q, \Sigma, \Omega, R, s, F)$ je líný konečný převodník. Pak konfigurací nazveme libovolný řetězec $\chi = vpu$, kde $p \in Q, u \in \Sigma^*, v \in \Omega^*$.

2.2.3.3 Definice přímého přechodu

Nechť $M = (Q, \Sigma, \Omega, R, s, F)$ je líný konečný převodník, nechť $\chi = vpxu$, $\chi' = vyqu$ jsou konfigurace, nechť $r: px \rightarrow yq \in R$, kde $p, q \in Q, x \in \Sigma^*, y \in \Omega^*$ je pravidlo. Pak říkáme, že M provede přímý přechod z χ do χ' podle pravidla $r: px \rightarrow yq \in R$ a zapisujeme $\chi \Rightarrow \chi'\,[r]$ nebo zkráceně $\chi \Rightarrow \chi'$.

2.2.3.4 Definice sekvence přímých přechodů

Nechť $M = (Q, \Sigma, \Omega, R, s, F)$ je líný konečný převodník.

- Nechť χ je konfigurace, pak M provede *0-přechodů* z χ do χ, zapisujeme $\chi \Rightarrow^0 [\varepsilon]$ nebo také zkráceně $\chi \Rightarrow^0 \chi$.
- Nechť $\chi_0, \chi_1, \ldots, \chi_n$ jsou konfigurace, kde $n \geq 1$ a $\chi_{i-1} \Rightarrow \chi_i\,[r_i]$, $r_i \in R$ pro všechna $i = 1, \ldots, n$. Pak říkáme, že M provede *n-přechodů* z χ_0 do χ_n a zapisujeme $\chi_0 \Rightarrow^n \chi_n\,[r_1 \ldots r_n]$ nebo také zkráceně $\chi_0 \Rightarrow^n \chi_n$.
- Nechť $\chi_0 \Rightarrow^n \chi_n\,[\rho]$ pro nějaké $n \geq 1$, pak říkáme, že χ_0 *netriviálně derivuje* χ_n a zapisujeme $\chi_0 \Rightarrow^+ \chi_n\,[\rho]$ nebo také zkráceně $\chi_0 \Rightarrow^+ \chi_n$.
- Nechť $\chi_0 \Rightarrow^n \chi_n\,[\rho]$ pro nějaké $n \geq 0$, pak říkáme, že χ_0 *derivuje* χ_n a zapisujeme $\chi_0 \Rightarrow^* \chi_n\,[\rho]$ nebo také zkráceně $\chi_0 \Rightarrow^* \chi_n$.

2.2.3.5 Definice překladu pomocí líného konečného převodníku

Nechť $M = (Q, \Sigma, \Omega, R, s, F)$ je líný konečný převodník. Překlad definovaný líným konečným převodníkem M (budeme jej označovat $T(M)$) je definován následovně:

$$T(M) = \{(x, y) : sx \Rightarrow^* yf, x \in \Sigma^*, y \in \Omega^*, f \in F\}.$$

2.2.4 Speciální typy konečných převodníků

V předchozí kapitole byl nadefinován líný konečný převodník. V následujících kapitolách bude ukázáno, že v praxi je vhodné použit pouze speciální líné konečné převodníky splňující jisté vlastnosti. Proto jsou v této kapitole nadefinovány tyto speciální typy líných konečných převodníků.

2.2.4.1 Definice jednoznačného líného konečného převodníku

Nechť $M = (Q, \Sigma, \Omega, R, s, F)$ je líný konečný převodník. Pak říkáme, že líný konečný převodník je *jednoznačný*, pokud pro každé $(x, y) \in T(M)$ existuje právě jedna posloupnost pravidel ρ taková, že: $sx \Rightarrow^* yf\,[\rho], f \in F$.

2.2.4.2 Definice konečného převodníku

Konečný převodník je šestice $M = (Q, \Sigma, \Omega, R, s, F)$, kde:

- Q je konečná množina stavů,
- Σ je vstupní abeceda,
- Ω je výstupní abeceda,
- R je konečná množina pravidel tvaru $pa \rightarrow yq$, kde $p, q \in Q, a \in \Sigma \cup \{\varepsilon\}, y \in \Omega^*$,
- $s \in Q$ je počáteční stav,
- $F \subseteq Q$ je množina koncových stavů.

Definice konfigurace, přímého přechodu, sekvence přechodů a definice překladu jsou stejné jako u líného konečného převodníku.

2.2.4.3 Definice konečného převodníku bez ε–přechodů

Konečný převodník bez ε–přechodů je šestice $M = (Q, \Sigma, \Omega, R, s, F)$, kde:

- Q je konečná množina stavů,
- Σ je vstupní abeceda,
- Ω je výstupní abeceda,
- R je konečná množina pravidel tvaru $pa \rightarrow yq$, kde $p, q \in Q - F, a \in \Sigma, y \in \Omega^*$ nebo $p \rightarrow yf$, kde $p \in Q - F, y \in \Omega^*$.
- $s \in Q$ je počáteční stav,
- $F = \{f\}$, přičemž $f \in Q$, je (jednoprvková) množina koncových stavů.

Definice konfigurace, přímého přechodu, sekvence přechodů a definice překladu je stejná jako u líného konečného převodníku.

2.2.4.4 Definice deterministického konečného převodníku

Deterministický konečný převodník M je konečný převodník bez ε−přechodů, pro který platí: pro všechna $a \in \Sigma$ a $p \in Q$ existuje maximálně jedno pravidlo tvaru $pa \rightarrow yq \in R$, kde $q \in Q - F$, $y \in \Omega^*$ a pro všechna $p \in Q$ existuje maximálně jedno pravidlo tvaru $p \rightarrow yf \in R$, kde $f \in F, y \in \Omega^*$.

Poznámka:

Zavedená definice se liší od standardní definice deterministického konečného převodníku, která je například uvedena v [4]. Tato modifikace je provedena kvůli korespondenci k implementovanému modelu.

2.2.5 Párové automaty

2.2.5.1 Definice párového konečného automatu

Párový konečný automat je trojice $\Gamma = (M_1, M_2, h)$, kde:

- $M_i = (Q_i, \Sigma_i, R_i, s_i, F_i)$ je líný konečný automat pro $i \in \{1, 2\}$,
- h je bijektivní zobrazení z R_1 do R_2.

Poznámka:

Zobrazení h můžeme rozšířit na zobrazení h^*, které definujeme následovně:

- $h^*(\varepsilon) = \varepsilon$
- pro $r_1, r_2, \ldots, r_n \in R_1$: $h^*(r_1 r_2 \ldots r_n) = h(r_1)(r_2)\ldots(r_n)$, kde $n \geq 1$.

2.2.5.2 Definice překladu pomocí párového konečného automatu

Nechť $\Gamma = (M_1, M_2, h)$ je párový konečný automat. Překlad definovaný párovým konečným automatem Γ (budeme jej označovat $T(\Gamma)$) je definován následovně:

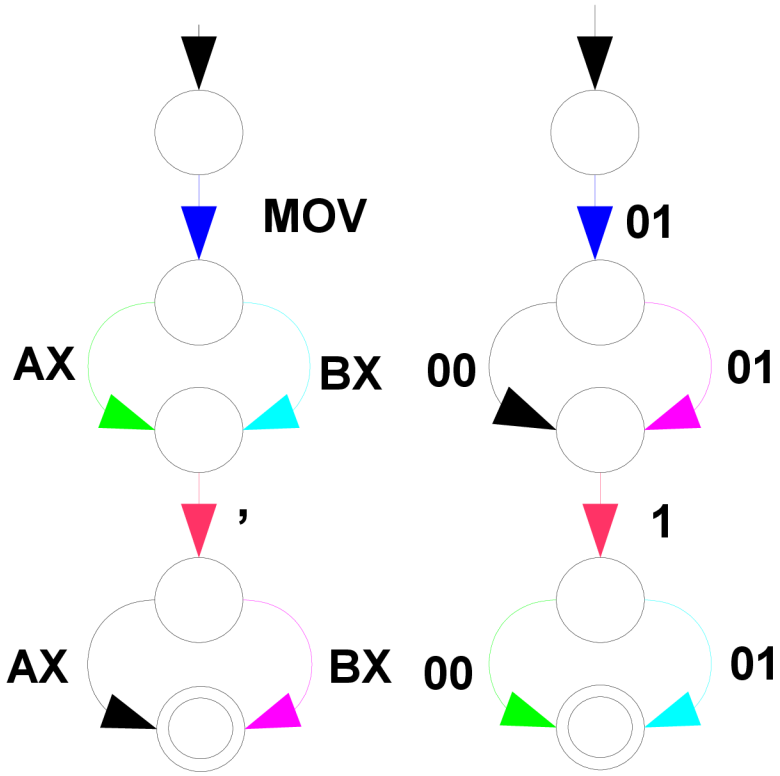
$T(\Gamma) = \{(w_1,w_2): w_1 \in \Sigma_1^*, w_2 \in \Sigma_2^*, s_1 w_1 \Rightarrow^* f_1\ [\rho_1]$ v $M_1, s_2 w_2 \Rightarrow^* f_2\ [\rho_2]$ v $M_2, f_1 \in F_1, f_2 \in F_2$,

$\rho \in perm(\rho_1), \rho_2 = h^*(\rho)\}$.

Poznámka:

perm(ρ) značí množinu všech řetězců, které vzniknou permutací symbolů v řetězci ρ.

2.2.5.3 Příklad párového konečného automatu



Obr. 1: Párový konečný automat

3 Determinizace líných konečných převodníků

Hlavním úkolem této práce je navrhnout schéma syntaktického analyzátoru pro efektivní překlad párového konečného automatu. Činnost párového automatu můžeme rozdělit do dvou fází:

1. fáze – provedení syntaktické analýzy vstupního řetězce vstupním automatem a podle toho, která pravidla byla použita, nastavení řízení výstupního automatu.

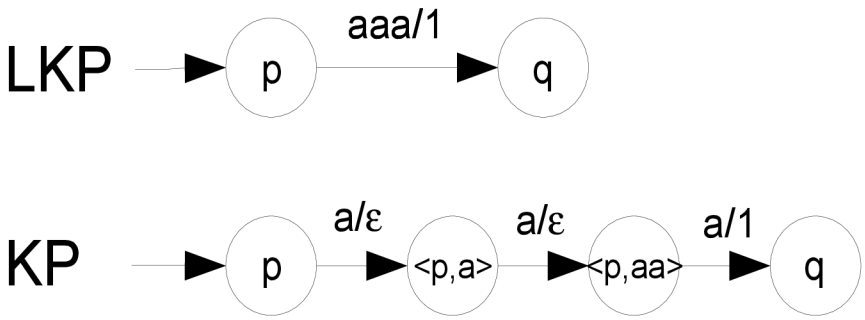
2. fáze – nagenerování výstupního řetězce výstupním automatem řízené nastavením řízení provedeném v 1. fází.

Aby 1. fáze mohla proběhnout deterministicky, je potřeba mít vstupní automat deterministický. Obecně tento automat ale deterministický být nemusí, proto je potřeba provést jeho determinizaci. Existuje spousta algoritmů, které transformují daný konečný automat na deterministický, přičemž ale zásadním způsobem změní strukturu daného automatu. Tím ztratíme informaci o tom, která pravidla původního automatu byla použita, což je zásadním problémem, neboť právě znalost sekvence použitých pravidel řídí činnost výstupního automatu. Daný problém budeme tedy řešit následovně: Vstupní líný konečný automat převedeme na líný konečný převodník, který bude jako výstup generovat sekvenci použitých pravidel. Toho můžeme docílit tím způsobem, že každé pravidlo původního líného automatu zmodifikujeme tak, že navíc na výstup vygeneruje své návěští. Tím se dostáváme do problematiky konečných převodníků. Tento konečný převodník se pokusíme převést na deterministický, aniž by se změnil daný překlad. To nám umožní deterministickým způsobem vygenerovat stejnou sekvenci pravidel, pomocí které by byl přijat daný řetězec pomocí původního líného automatu. Proto se v této kapitole budeme zabývat determinizací líných konečných převodníků, neboť tyto převodníky později budou tvořit jádro našeho syntaktického analyzátoru. Tato determinizace proběhne celkem ve třech fázích, které jsou popsány v následujících kapitolách.

3.1 Převod líného konečného převodníku na konečný převodník

Níže uvedený algoritmus převádí líný konečný převodník (LKP) na konečný převodník (KP) tak, že se může zvýšit původní počet stavů a konečnému převodníku mohou být přidána nová pravidla. LKP při přechodu z jednoho stavu do druhého přijímá obecně *řetězec*. Oproti tomu KP může na vstupu přijmout nejvýše jeden symbol. Pokud má LKP následující pravidlo $px \rightarrow yq$ přičemž $|x| \geq 1$, tedy při přechodu přečteme více než jeden symbol, postupujeme následovně. Na vstupu KP přečteme pouze jeden symbol řetězce x a přejdeme do nově vytvořeného stavu, přičemž na výstup se nic nezapíše.

Nově vzniklý stav pojmenujeme tak, že název bude obsahovat stav *p*, který je výchozím stavem a první přečtený symbol. Pokud např.: $x = aaa$, tak název nového stavu bude $<p, a>$, další bude $<p, aa>$. Na výstup se bude zapisovat až po přečtení posledního symbolu řetězce *x* a přejde se do stavu, do kterého by se přešlo pomocí LKP po přečtení řetězce *x*. Pokud $|x| = 1$ provedeme normální přechod do následujícího stavu. Tento postup je znázorněn na obr.2.



Obr. 2: Převod LKP na KP

3.1.1 Formální algoritmus pro převod líného konečného převodníku na konečný převodník

Vstup: LKP $M_1 = (Q_1, \Sigma, \Omega, R_1, s, F)$

Výstup: KP $M_2 = (Q_2, \Sigma, \Omega, R_2, s, F)$ takový, že $T(M_1) = T(M_2)$

Metoda: $Q_2 := Q_1$;

for each $px \to yq \in R_1$ **do**

if $|x| \leq 1$ **then**

add $px \to yq$ **to** R_2

else

let $x = a_1a_2 \ldots a_n$, where $n \geq 2$:

add $<p, a_1>, <p, a_1a_2>, \ldots, <p, a_1a_2 \ldots a_{n-1}>$ **to** Q_2

add $pa_1 \to <p, a_1>, <p, a_1>a_2 \to <p, a_1a_2>, \ldots, <p, a_1a_2 \ldots a_{n-2}>a_{n-1} \to <p, a_1a_2 \ldots a_{n-1}>,$

$<p, a_1a_2 \ldots a_{n-1}>a_n \to yq$ **to** R_2

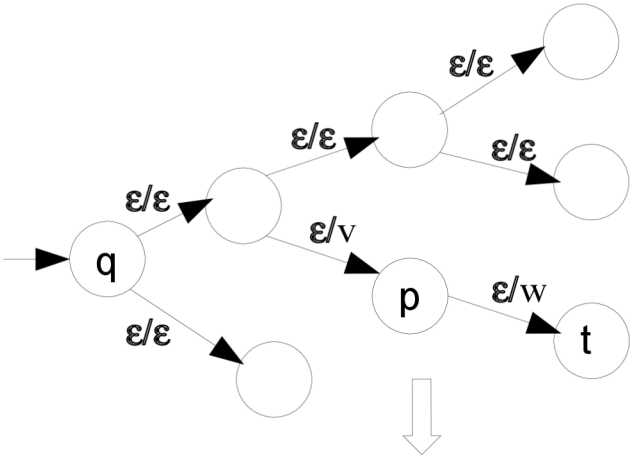
3.2 Převod konečného převodníku na konečný převodník bez ε–přechodů

3.2.1 Definice ε–uzávěru

Nechť $M = (Q, \Sigma, \Omega, R, s, F)$ je jednoznačný líný konečný převodník a $q \in Q$. ε–uzávěr pak definujeme jako:

$$\varepsilon\text{–uzávěr}(q) = \{(p, y): q \Rightarrow^* yp, \text{kde } p \in Q, y \in \Omega^*\}$$

Na následujícím obrázku je provedena ilustrace této definice.



$(t, vw) \in \varepsilon\text{–uzávěr}(q)$

Obr. 3: Příklad na ε–uzávěr

3.2.2 Výpočet ε–uzávěru

ε–uzávěr budeme potřebovat při převodu konečného převodníku (KP) na KP bez ε–přechodů. Budeme jej počítat pomocí následujícího algoritmu. Mějme dvě dvě množiny S_{done} a S_{undone}. Obě množiny obsahují dvojice stav - sémantická akce. Na začátku množině S_{undone} přiřadíme prvek (q, ε), kde $q \in Q$ je stav, u kterého počítáme ε–uzávěr. Postupně z této množiny bereme prvky dokud není prázdná. Jako první zkontrolujeme, zda v množině S_{done} neexistuje jiný prvek obsahující stejný stav, který je ale vygenerovaný jinou posloupností sémantických akcí. Pokud taková dvojice existuje, nastává chyba nejednoznačnosti. V opačném případě přiřadíme tento prvek do množiny S_{done} a zároveň jej odstraníme z množiny S_{undone}. Dále chceme zjistit, zda v automatu existuje stav t, do kterého se lze dostat ze stavu p vygenerováním sémantické akce w. Pokud taková dvojice $<t, w>$ existuje, přidáme ji do množiny S_{undone}.

3.2.3 Formální algoritmus pro výpočet ε–uzávěru

Vstup: LKP $M = (Q, \Sigma, \Omega, R, s, F)$, $q \in Q$

Výstup: ε–uzávěr(q) nebo chyba detekující nejednoznačnost M

Metoda: $S_{undone} := \{(q, \varepsilon)\}$;

$S_{done} := \emptyset$;

while $S_{undone} \neq \emptyset$ **do begin**

let $(p, v) \in S_{undone}$:

if exists $w \in \Omega^*$ **such that** $(p, w) \in S_{done}$ **then** error(ambiguity)

$S_{undone} := S_{undone} - \{(p, v)\}$;

$S_{done} := S_{done} \cup \{(p, v)\}$;

$S_{undone} := S_{undone} \cup \{(t, vy): p \rightarrow yt \in R\}$;

end

ε–uzávěr(q) := S_{done};

3.2.4 Převod konečného převodníku na konečný převodník bez ε–přechodů

Abychom mohli vytvořit konečný převodník bez ε–přechodů, potřebujeme znát konečný převodník a ε–uzávěry všech jeho stavů. Oba převodníky vytvářejí stejný překlad. Na začátku přidáme do množiny stavů převodníku stavy z původního převodníku a přidáme zde i koncový stav tohoto

převodníku. Množina pravidel zatím zůstane prázdná. Pak postupně procházíme všechny stavy q, pro které spočítáme ε–uzávěr. Pokud z nějakého stavu p z tohoto ε–uzávěru přečtením symbolu a a nagenerováním řetězce y můžeme přejít do stavu t, přidáme do množiny pravidel nové pravidlo tvaru $qa \rightarrow xyt$, kde x je výstupní řetězec přidružený stavu p v ε–uzávěru.

Nakonec přidáme ještě jedno pravidlo, pomocí kterého převodník nic nepřečte na vstupu a přejde do nově vytvořeného koncového stavu f_2. Toto pravidlo je přidáno z toho důvodu, že pokud bychom přešli do původního koncového stavu pomocí ε, nemohli bychom nagenerovat výstupní řetězec celý, což by jistě vedlo k problémům při dalším zpracování.

3.2.4.1 Formální algoritmus pro převod konečného převodníku na konečný převodník bez ε–přechodů

Vstup: KP $M_1 = (Q_1, \Sigma, \Omega, R_1, s, F_1)$, ε–uzávěr($q$) pro všechna $q \in Q$

Výstup: KP bez ε–přechodů $M_2 = (Q_2, \Sigma, \Omega, R_2, s, F_2)$ takový, že $T(M_1) = T(M_2)$.

Metoda: $Q_2 := Q_1 \cup \{f_2\}$;

$R_2 := \varnothing$;

$F_2 := \{f_2\}$;

for each $q \in Q$ **do**

$R_2 := R_2 \cup \{qa \rightarrow xyt: pa \rightarrow yt \in R_1, (p, x) \in$ ε–uzávěr(q), $a \in \Sigma, t \in Q\}$

$\cup \{q \rightarrow xf_2: (f, x) \in$ ε–uzávěr(q), $f \in F\}$;

3.3 Převod konečného převodníku bez ε–přechodů na deterministický konečný převodník

Nejjednodušším případem je, že ze stavu přejdeme přečtením symbolu do právě jednoho nového stavu. V tomto případě přečteme vstupní symbol a provedeme danou sémantickou akci. V novém stavu si nemusíme uchovávat žádné informace kromě názvu nového stavu. Tento stav vyjádříme jako dvojici <nový_stav, ε >.

Druhým případem je tato situace: Z právě jednoho stavu můžeme přejít do 2 nebo více nových stavů přečtením stejného symbolu. Tato situace se řeší tak, že přejdeme do nového stavu, který

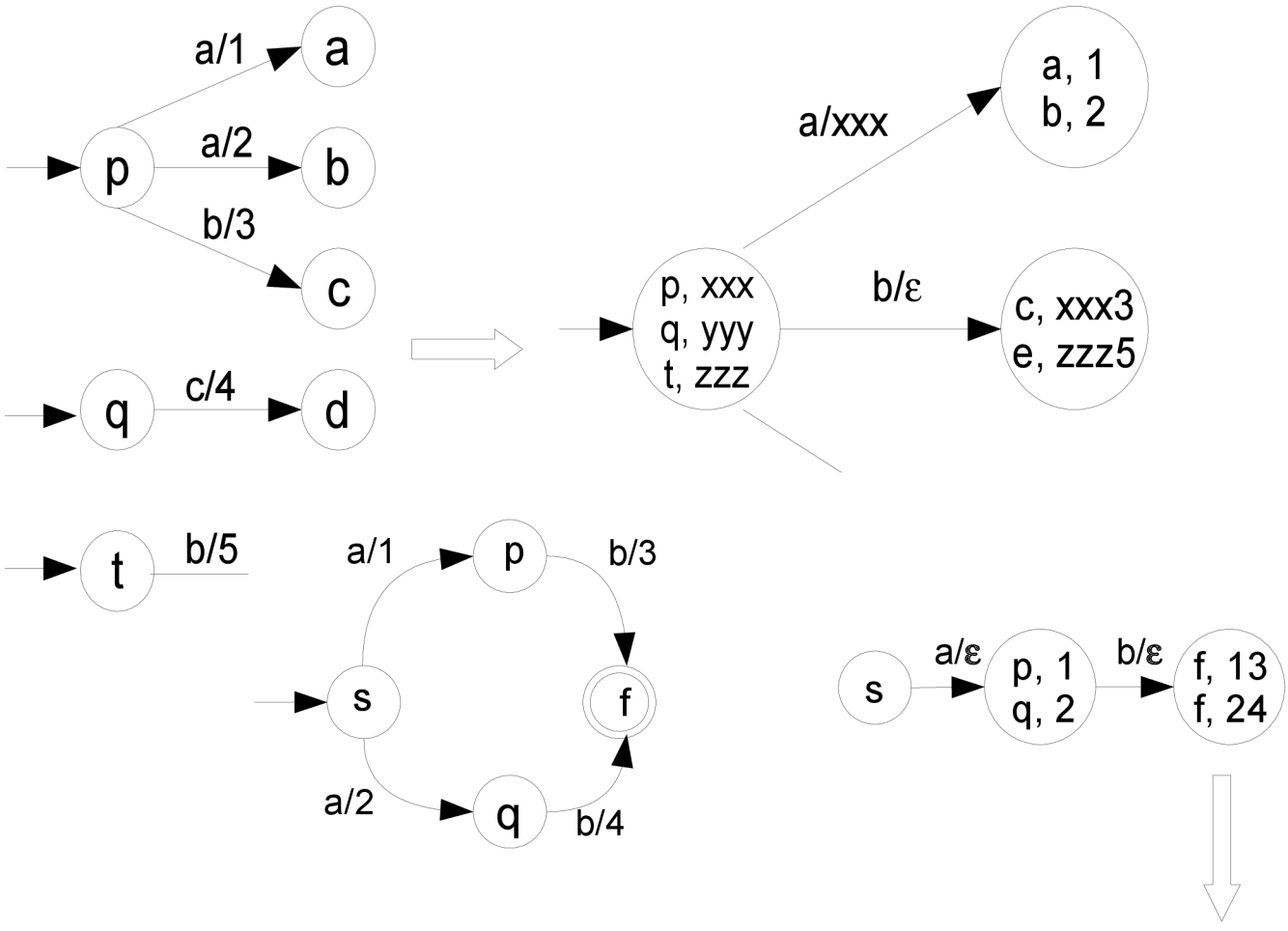
obsahuje množinu dvojic <nový stav, sémantická akce, která by se při přechodu měla provést>. Při přechodu provedeme sémantické akce, které byly případně uchovány v původním stavu.

Třetí a zároveň nejsložitější situací je, pokud čtením právě jednoho stejného symbolu můžeme přejít ze dvou různých stavů do jiných stavů. Hraně nastavíme jako vstupní řetězec daný symbol, přičemž sémantická akce se nevykoná žádná. V nově vzniklém stavu uchováváme dvojice <nový stav, sémantická akce>. Sémantická akce je složena ze sémantické akce uchované v původním stavu (pokud ji tento obsahoval) a sémantické akce, která by se vykonala při přečtení vstupního symbolu.

Na následujícím obrázku (obr.2) jsou znázorněny všechny tři předchozí případy. První případ nastane, pokud máme na vstupu přečíst symbol *c*. Přečtením tohoto symbolu můžeme přejít pouze do stavu *d*. Při přechodu se provede sémantická akce, která byla uložena u původního stavu *q*, ze kterého hrana vychází a také sémantická akce náležící přechodu po přečtení symbolu *c*.

Druhá situace nastane čtením symbolu *a*. Protože obě hrany vychází ze stejného stavu, vykoná se při přechodu sémantická akce uložená u původního stavu a v novém stavu budou dvojice nových stavů a nevykonaných sémantických akcí, tedy <*a*, 1>, <*b*, 2>.

A konečně přečtením symbolu *b* nastane nejsložitější případ. Při přechodu se na výstup nic nezapíše, ale v novém stavu musíme uchovat oba stavy *c* a *e* ve dvojicích s řetězci sémantických akcí z původního stavu a vykonávaného přechodu, tedy *xxx*3 a *zzz*5.



Obr. 5: Příklad na nejednoznačnost

3.3.1 Formální algoritmus pro převod konečného převodníku bez ε–přechodů na deterministický konečný převodník

Vstup: KP bez ε–přechodů $M_1 = (Q_1, \Sigma, \Omega, R_1, s_1, \{f_1\})$

Výstup: DKP $M_2 = (Q_2, \Sigma, \Omega, R_2, s_2, \{f_2\})$ nebo chyba, pokud je M_1 nejednoznačný nebo pokud převod provést tímto algoritmem nelze

Metoda: $s_2 := \{<s_1, \varepsilon>\}$; $f_2 := \{<f_1, \varepsilon>\}$;

$S_{undone} := \{s_2\}$;

$S_{done} := \varnothing$;

$S_{cycle} := \{(s_2, \varepsilon)\}$;

while $S_{undone} \neq \varnothing$ **do begin**

let $X \in S_{undone}$:

$S_{undone} = S_{undone} - \{X\}$;

$S_{done} := S_{done} \cup \{X\}$;

for each $a \in \sum \cup \{\varepsilon\}$ **do**

$Q_{in} := \{<p, z>: <p, z> \in X, pa \rightarrow yq \in R_1, y, z \in \Omega^*, q \in Q\}$;

if $|Q_{in}| = 1$ **then**

let $<p, z> \in Q_{in}$:

$Q_{out} := \{<q, y>: q \in Q, y \in \Omega^*, pa \rightarrow yq \in R_1\}$;

if $|Q_{out}| = 1$ **then**

let $<q, y> \in Q_{out}$:

$Q_{out} := \{<q, \varepsilon>\}$;

add $Xa \rightarrow zyQ_{out}$ **to** R_2;

if $|Q_{out}| > 1$ **then**

add $Xa \rightarrow zQ_{out}$ **to** R_2;

if $|Q_{in}| > 1$ **then**

$Q_{out} := \{<q, zy>: <p, z> \in X, pa \rightarrow yq \in R_1, y, z \in \Omega^*, q \in Q\}$;

add $Xa \rightarrow Q_{out}$ **to** R_2;

$S_{undone} = S_{undone} \cup \{Q_{out}\}$;

if exists $q \in Q$ **such that** $|\{y: <q, y> \in Q_{out}\}| > 1$ **then** error(ambiguity)

let $(X, x) \in S_{cycle}$: **if exists** $(Y, y) \in S_{cycle}$ **such that** y is a prefix of x, $X \neq Y$,

$\{p: <p, u> \in X\} = \{q: <q, v> \in Y\}$ **then** error(cycle)

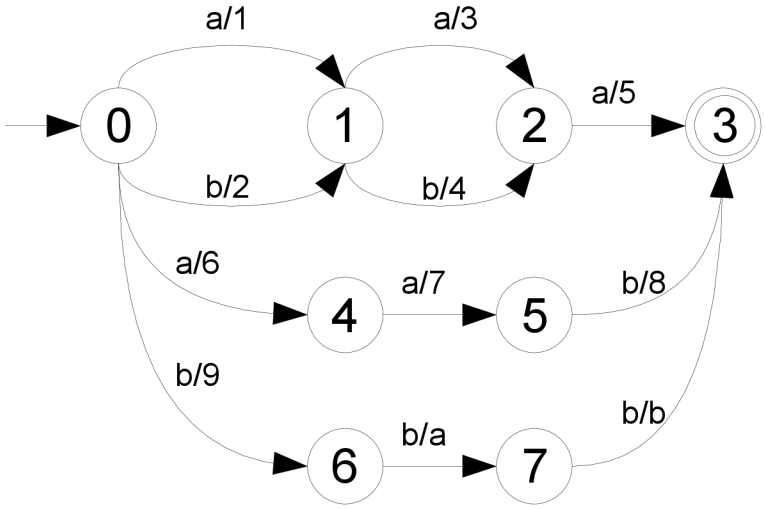
add (Q_{out}, xa) **to** S_{cycle}

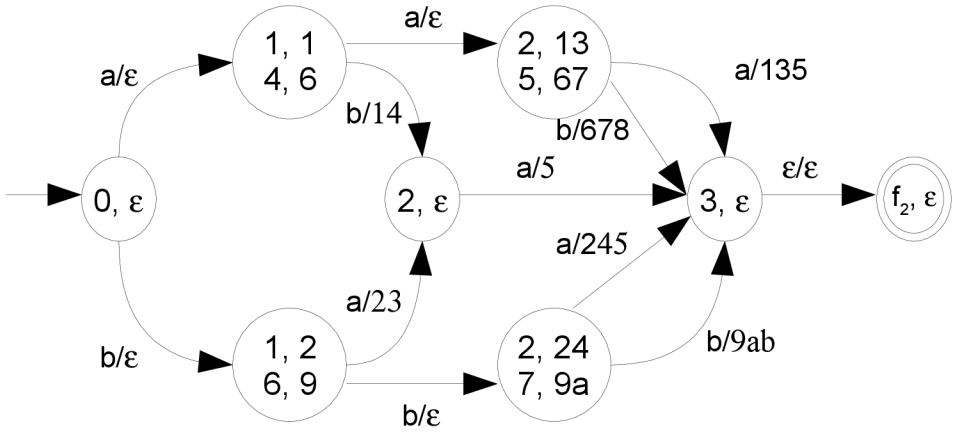
end

$Q_2 := S_{done}$;

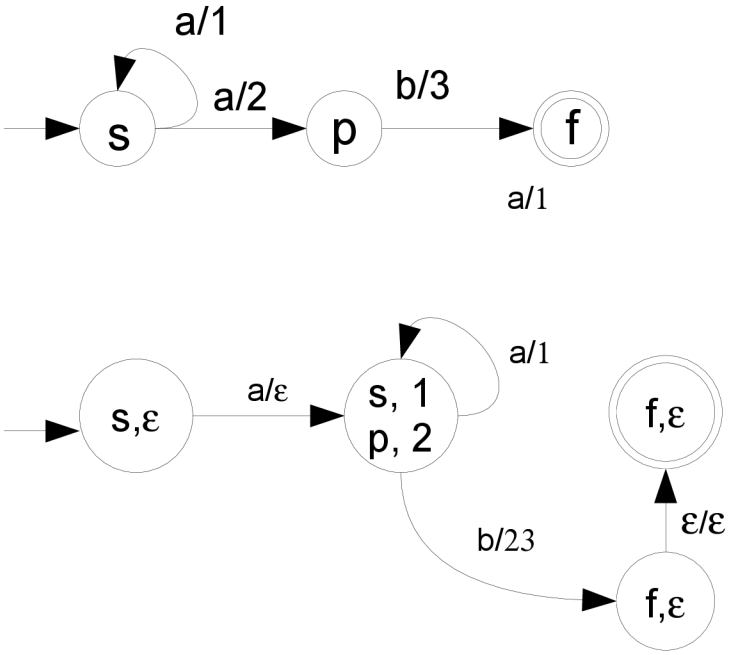
3.3.1.1 Příklad převodu

Činnost předchozího algoritmu je ilustrována následujícími obrázky ve dvou různých příkladech.





Obr. 6: Příklad: převod na deterministický automat



Obr. 7: Příklad

4 Princip činnosti překladače BISON--

YACC je jazyk a překladač, který umožňuje automaticky vytvořit syntaktický analyzátor založený na LR-syntaktické analýze na základě LR-gramatiky, kterou si uživatel sám nadefinuje. Pro nekomerční použití vznikl překladač BISON, který je z hlediska svých schopností srovnatelný s YACC. Pro naše účely je ale tento překladač nevhodný, protože při syntaktické analýze využívá složitých LR-tabulek a zásobníku, což velice zpomaluje jeho činnost. Proto tento překladač zjednodušíme pro naše účely. Tento překladač bude mít název BISON--. Zjednodušení bude především v tom, že od uživatele bude vyžadována pouze pravá lineární gramatika. Toto zjednodušení povede k vyšší rychlosti překladu, neboť v průběhu překladu nebude použit zásobník.

4.1 Jazyk BISON--

V této sekci popíšeme syntax a sémantiku jazyku BISON--. Základní verze definuje pouze popis pravé lineární gramatiky se sémantickými akcemi.

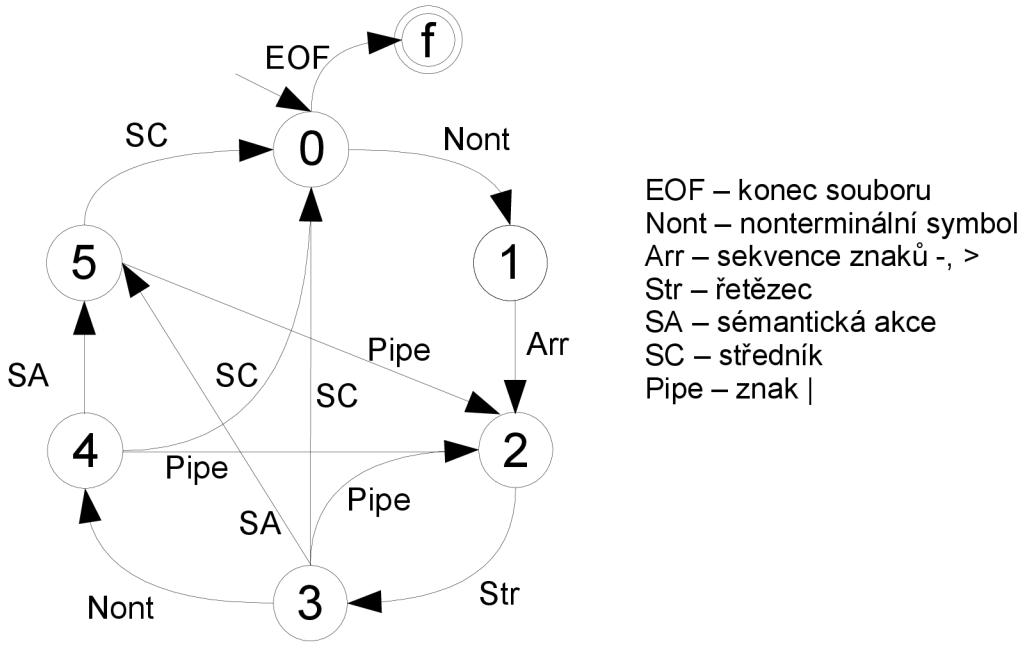
Pravidla ve tvaru

```
Nont_0 -> "Ret_1" Nont_1 {Sem_akce_1}
Nont_0 -> "Ret_2" Nont_2 {Sem_akce_2}
      ...
Nont_0 -> "Ret_n" Nont_n {Sem_akce_n}
```

jsou v jazyce BISON-- zapsány jako

```
Nont_0 -> "Ret_1" Nont_1 {Sem_akce_1}
      |   "Ret_2" Nont_2 {Sem_akce_2}
      ...
      |   "Ret_n" Nont_n {Sem_akce_n}
      ;
```

přičemž nonterminální symboly na pravé straně pravidla a sémantické akce jsou nepovinné. Program v jazyce BISON-- obsahuje sekvenci takových konstrukcí. Nonterminální symbol na levé straně prvního pravidla se bere jako počáteční. Sémantika daného pravidla je následující: Pokud je během syntaktické analýzy použito pravidlo, které obsahuje sémantickou akci, je tato sémantická akce vykonána. Syntax jazyka BISON-- je formálně popsána pomocí konečného automatu v následujícím obrázku:



Obr. 8: Syntax jazyka BISON--

Poznamenejme, že pomocí pravé lineární gramatiky můžeme jednoduše odsimulovat činnost líného konečného automatu. Převod může být například proveden pomocí následujícího algoritmu.

4.1.1 Formální algoritmus pro převod líného konečného automatu na ekvivalentní pravou lineární gramatiku

Vstup: LKA $M = (Q, \Sigma, R, s, F)$

Výstup: PLG $G = (N, T, P, S)$ taková, že $L(M) = L(G)$

Metoda: $N := Q$;

$T := \Sigma$;

$s := S$;

$P := \{p \rightarrow xq: px \rightarrow q \in R\} \cup \{f \rightarrow \varepsilon: f \in F\}$

4.2 Implementace BISON--

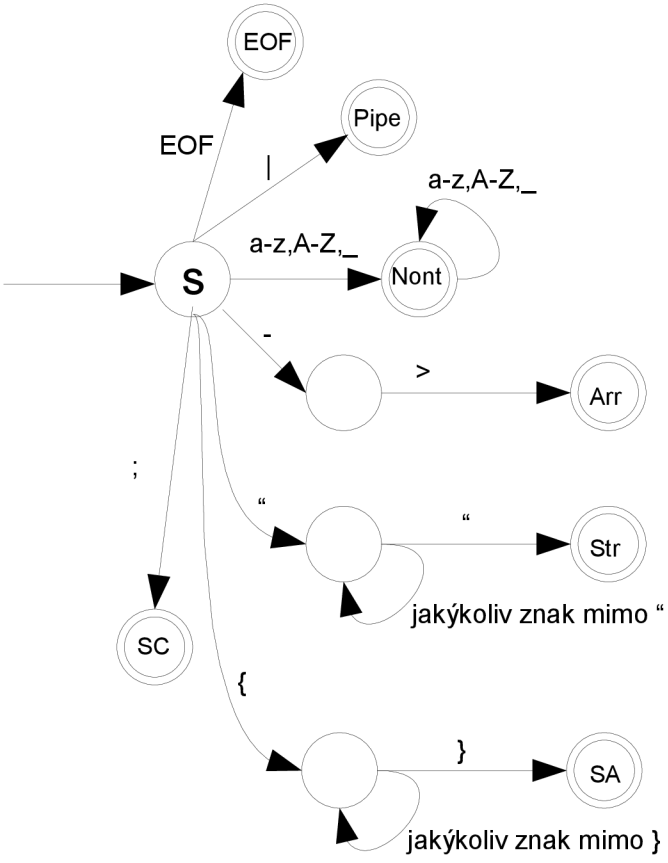
V základní verzi popíšeme tvorbu syntaktických analyzátorů pomocí tohoto programu. V následující kapitole bude popsáno i její rozšíření na provedení překladu pomocí párových konečných automatů. Tento program je implementován v jazyce C++. Stěžejní komponentou je třída tAutomaton, která obsahuje následující veřejné metody:

- read() - načte ze standardního vstupu strukturu líného konečného převodníku reprezentovaného pravou lineární gramatikou se sémantickými akcemi v jazyce BISON--.
- write() - vypíše na standardní výstup strukturu líného konečného převodníku v takovém formátu, aby mohl být zpracován programem DOT, který automaticky vytvoří grafickou reprezentaci daného automatu.
- convertToDeterministic() - pokusí se převést daný líný konečný převodník na ekvivalentní deterministický.
- setEdge() - vytvoří novou hranu k danému automatu
- generateParser() - vygeneruje program v jazyce C, který již obsahuje implementaci syntaktického analyzátoru, který bude simulovat načtený líný konečný převodník.
- clear() - zruší všechny hrany líného konečného převodníku.

Implementační detaily složitějších metod jsou popsány v následujících podkapitolách.

4.2.1 Implementace metody read()

Tato metoda načte ze standardního vstupu strukturu líného konečného převodníku reprezentovaného pravou lineární gramatikou se sémantickými akcemi v jazyce BISON--. Je realizována pomocí lexikálního a syntaktického analyzátoru překladače. Struktura lexikálního analyzátoru je zobrazena na následujícím obrázku.



Obr. 9: Struktura lexikálního analyzátoru

Činnost syntaktického analyzátoru je založena na konečném automatu ilustrovaném v předchozí kapitole.
V průběhu načítání hran již přiřazuje názvům jednotlivých stavům číselné hodnoty z důvodu optimalizace.

4.2.2 Implementace metody convertToDeterministic()

Tato metoda se pokusí převést daný líný konečný převodník na ekvivalentní deterministický. Toto je prováděno ve třech etapách:

- I. etapa: Líný konečný převodník je převeden na ekvivalentní konečný převodník. Tato fáze je provedena podle algoritmu 3.1.1. Výstupem této etapy je vždy konečný převodník, neboť tato transformace je vždy možná.
- II. etapa: Konečný převodník je převeden na konečný převodník bez ε-přechodů. Tato fáze je provedena podle algoritmu 3.2.4. Tato etapa nemusí vždy skončit úspěchem, neboť již v rámci ε-přechodů může dojít k nejednoznačnosti automatu, což je zachyceno v této fázi. Pokud tomu tak je, uživateli se vrací kód příslušné chyby a ve struktuře zůstane původní líný konečný převodník.
- III. etapa: Konečný převodník bez ε-přechodů je převeden na deterministický konečný převodník. Tato fáze je provedena podle algoritmu 3.3.1. Tato etapa nemusí vždy skončit úspěchem, neboť může být detekována nejednoznačnost, a nebo tato konverze výše uvedeným algoritmem není možná. Pokud tomu tak je, uživateli se vrací kód příslušné chyby a ve struktuře zůstane původní líný konečný převodník.

4.2.3 Implementace metody generateParser()

Tato metoda vygeneruje program v jazyce C, který obsahuje již implementaci syntaktického analyzátoru, který bude simulovat načtený líný konečný převodník. Tento kód v jazyce C se generuje na standardní výstup. Jedná se o vygenerování funkce parse(), jejíž tělo obsahuje kód tohoto syntaktického analyzátoru. Základem struktury je přepínač (switch) v závislosti na aktuálním stavu, jehož jednotlivé sekce obsahují další přepínač v závislosti na načteném aktuálním symbolu. Tyto sekce již odsimulují dané pravidlo líného konečného automatu. A to tak, že nastaví aktuální stav na nový a vykonají přidruženou sémantickou akci. Poté je ze vstupu načten další symbol. Toto načítání je prováděno tak dlouho, dokud není načten symbol značící konec souboru (EOF) nebo dokud je možné v daném líném konečném převodníku provádět přechody. Pokud s novým symbolem již nejde provést přechod, nasimuluje se situace, jako by k načtení symbolu nedošlo. Vrací se příslušný chybový kód, tj. jestli proběhla část syntaktické analýzy v pořádku (konečný převodník došel do

koncového stavu) a jestli byla ukončena koncem souboru nebo nemožností provedení dalšího přechodu.

4.3 Rozšíření BISON-- pro simulaci překladu pomocí párových automatů

V této kapitole je uvedeno rozšíření jazyka Bison pro simulaci párových konečných automatů. Párový konečný automat byl nadefinován v kapitole 2. Z hlediska zefektivnění syntaktického analyzátoru budeme párovat v daných automatech místo hran stavy. Je to z důvodu zrychlení daného algoritmu. Párovost stavů provedeme tak, že u obou automatů budou oba stavy tvořící pár identicky pojmenovány.

Rozšíření z hlediska syntaxe je následující:

```
Popis_vstupniho_automatu
%%
Popis_vystupniho_automatu
%%
Pomocné deklarace a funkce
```

Sekce **Popis_vstupniho_automatu** obsahuje popis konečného převodníku tak, jak bylo uvedeno v základní verzi BISON--. Pomocí sémantických akcí jsou tu ale nastavovány různé globální proměnné na jisté hodnoty, které potom budou řídit generování prováděné výstupním automatem. Pro každý stav s názvem např. **xxx** je tu virtuálně vytvořena globální proměnná s názvem **switch_xxx**. Do této proměnné můžeme přiřadit název libovolného stavu. Tato sémantická akce bude tedy např. ve tvaru:

```
switch_xxx := yyy
```

Význam této sémantické akce je následující: Pokud se výstupní automat bude nacházet ve stavu **xxx**, provede se přechod právě podle té hrany, která vede do stavu **yyy**.

Sekce **Popis_vystupniho_automatu** obsahuje popis ve stejném formátu, jako bylo v předchozí sekci pro vstupní automat, ale bude se týkat výstupního automatu. Tento automat bude sloužit pro generování výstupu. Pokud z aktuálního stavu vede pouze jedno pravidlo, provede se přechod pomocí tohoto pravidla a vygeneruje se na výstup patřičný řetězec. Pokud z aktuálního stavu **xxx** vede více pravidel, vybere se právě to pravidlo, které vede do stavu, na který se odkazuje proměnná **switch_xxx**.

Sekce **Pomocné deklarace a funkce** obsahuje pomocné funkce, které mohou být volány v rámci sémantických akcí. Dále tu musí být uvedeny deklarace všech proměnných tvaru **switch_xxx** jako integer. Tato sekce bude později pouze zkopírována do daného kódu.

4.3.1 Implementace daného rozšíření

Třída tAutomaton navíc obsahuje metodu generateGenerator(), která vygeneruje kód v jazyce C simulující činnost výstupního konečného automatu. Pokud daný automat obsahuje z aktuálního stavu pouze jedno pravidlo, je proveden přechod po této hraně. Pokud je ovšem těchto pravidel více, rozhodne příslušná proměnná **switch_xxx** o tom, kterou hranou se bude pokračovat v generování.

4.3.2 Příklad

Tento příklad provede překlad symbolu 0 na 1 a symbolu 1 na 0.

```
start -> "" jednicka { switch_start = jednicka; }
      |  "" nula     { switch_start = nula; }
      ;
jednicka -> "1" konec
      ;
nula -> "0" konec
      ;
konec -> ""
      ;
%%
start -> "" jednicka
      |  "" nula
      ;
jednicka -> "1" konec
      ;
nula -> "0" konec
      ;
konec -> ""
      ;
%%
int switch_start;
```

5 Závěr

Tato bakalářská práce se zabývala systémem modelů líných konečných automatů, které simulovaly překlad. Nejobtížnější pasáží byla právě determinizace vstupního automatu. V této bakalářské práci byly navrženy algoritmy, které vhodným způsobem tuto transformaci prováděly, aniž bychom ztratili informaci o použitých pravidlech. V práci bylo ukázáno, že tento problém nám řeší determinizace líného konečného převodníku. Z teorie formálních jazyků je obecně známo, že existují takové líné konečné převodníky, které nelze převést na deterministické. Z toho důvodu nelze tento algoritmus aplikovat na libovolný líný konečný převodník. Dá se ovšem dokázat, že lze použít na libovolný líný konečný převodník provádějící překlad, jehož vstupní jazyk je konečný. Pokud tedy chceme například provádět rychlým způsobem překlad z binárního kódu do assembleru, můžeme této metody využít, neboť délka instrukce binárního kódu je vždy omezená jistou konstantou.

Teoretický přínos:

Bakalářská práce formálně popisuje algoritmy, které provádějí převod některých líných konečných převodníků na deterministické. Tyto algoritmy tedy mají uplatnění v teorii formálních jazyků.

Praktický přínos:

V rámci této bakalářské práce byl naimplementován překladač jazyka BISON--, který automaticky generuje kód v jazyce C simulující činnost syntaktického analyzátoru, který si uživatel popíše pomocí pravé lineární gramatiky. Na jedné straně se jedná pouze o zjednodušení klasického jazyka BISON, který může mít na vstupu obecně LR-gramatiku, ale na druhé straně automaticky vytvořený syntaktický analyzátor pracuje efektivněji a rychleji. V praxi se tento druh překladače osvědčil, neboť existuje spousta případů, ve kterých je právě vyžadována rychlost kompilátoru překládající jednoduchý jazyk. Příkladem je třeba rychlé generování assemblerovského kódu z binárního kódu za účelem simulování.

Literatura

[1] Beneš, M., Hruška, T, Češka, M.: Překladače, Brno: VUT, 1994.

[2] Lukáš, R., Hruška, T., Kolář, D., Masařík, K.: Two-Way Deterministic Translation and Its Usage in Practice. Proceedings of 8th Spring International Conference - ISIM'05, Ostrava, CZ, MARQ, 2005.

[3] Masařík, K., Hruška, T., Kolář, D., Lukáš R.: System for design and simulation of microprocessors. Proceedings of 8th Spring International Conference - ISIM'05, Ostrava, CZ, MARQ, 2005.

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

[5] Meduna, A.: Two-Way Metalinear PC Grammar Systems and Their Descriptional Complexity. Acta Cybernetica, 2003.

[6] Meduna, A., Lukáš, R.: Multigenerative Grammar Systems. Schedae Informaticae, Krakov, PL, 2006.

[7] Salomaa, A.: Formal Languages. Academic Press, New York, 1973.

Seznam příloh

Příloha 1. CD