Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DISERTAČNÍ PRÁCE

Martin Procházka

Redukční automaty a syntaktické chyby

Katedra teoretické informatiky a matematické logiky

Vedoucí disertační práce:	Martin Plátek, CSc.
Studijní program:	Informatika
Studijní obor:	I3 – Matematická lingvistika

Praha 2012

Děkuji Martinu Plátkovi, vedoucímu předkládané práce, za cenné připomínky, za to, že mi umožnil věnovat se problémům, jejichž řešení mi přinášelo zábavu a uspokojení, i za to, že se mu volnost, kterou mi po většinu mého doktorského studia poskytoval, nakonec podařilo usměrnit a dovést mě tak k dokončení celé práce.

Své ženě Kateřině jsem vděčný za zázemí, které celé naší rodině poskytuje, a za trpělivost, s jakou snášela dlouhé hodiny, které jsem po večerech a volných dnech věnoval této práci (zejména v posledních několika měsících).

Můj dík patří rovněž Grantové agentuře České republiky za poskytnutí grantu GAP202/10/1333, jehož přispěním mohla tato práce vzniknout.

Prohlašuji, že jsem tuto disertační práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Redukční automaty a syntaktické chyby

Autor: Martin Procházka

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí disertační práce: Martin Plátek, CSc., Katedra teoretické informatiky a matematické logiky

Abstrakt: Tato práce se zabývá *redukčními automaty*, jejich *normalizacemi* a využitím pro *(robustní) redukční analýzu* a *lokalizaci syntaktických chyb* pro jazyky ze třídy DCFL. Redukční automat navazuje na restartovací automat, od kterého se odlišuje explicitním určením redukovaných symbolů (to umožňuje přesné určení místa chyby) a přesunem výhledového okna do stavu řídící jednotky (ten ho přibližuje zařízením studovaným klasickou teorií automatů a formálních jazyků). Pro redukční automat pak lze snadněji než pro automat restartovací přebírat pojmy a postupy klasické teorie, jako je např. *prefixová korektnost* nebo *minimalizace množiny stavů*. Pro libovolný neprázdný jazyk ze třídy DCFL zadaný monotónním redukčním automatem, navíc ještě prefixově korektním a stavově minimálním, navrhujeme metodu robustní redukční analýzy, která zaručuje lokalizaci formálně definovaných typů skutečných (nezavlečených) chyb, bezchybných podslov a míst redukčních konfliktů (podslov s nejednoznačnou syntaktickou strukturou redukovatelných v různých slovech různými způsoby). Navrhovanou metodu implementujeme novým typem zařízení (postprefixovým robustním analyzátorem) a stručně ukazujeme, jak jí implementovat deterministickým zásobníkovým převodníkem pracujícím v lineárním čase vzhledem k délce slova.

Klíčová slova: redukční automaty, robustní redukční analýza, lokalizace syntaktických chyb

Title: Reducing automata and syntactic errors

Author: Martin Procházka

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Martin Plátek, CSc., Department of Theoretical Computer Science and Mathematical Logic

Abstract: This thesis deals with *reducing automata*, their *normalization*, and their application for *a (robust) reduction analysis* and *localization of syntactic errors* for deterministic context-free languages (DCFL). A reducing automaton is similar to a restarting automaton with two subtle differences: an explicit marking of reduced symbols (which makes it possible to determine a position of an error accurately), and moving a lookahead window inside a control unit (which brings reducing automata closer to devices of classical automata and formal language theory). In case of reducing automata, it is easier to adopt and reuse notions and approaches developed within classical theory, e.g., *prefix correctness* or *automata minimization*. For any nonempty deterministic context-free language specified by a monotone reducing automaton, both prefix correct and minimal, we propose a method of robust analysis by reduction which ensures localization of formally defined types of (real) errors, correct subwords, and subwords causing reduction conflicts (i.e., subwords with ambiguous syntactic structure that can be reduced in different words in different ways). We implement the proposed method by a new type of device (called postprefix robust analyzer) and we briefly show how to implement this method by a deterministic pushdown transducer working in linear time with respect to the length of the analyzed word.

Keywords: reducing automata, robust analysis by reduction, localization of syntactic errors

Obsah

Úvod **3**

1 Redukční automat **7**
1.1 Definice a základní vlastnosti 7
1.2 Reprezentace red-automatů 13
1.3 Shrnutí 15

2 Redukční analýza chybných slov **17**
2.1 Prefixové a postprefixové (ne)korektnosti 18
2.2 Normovaný mon-red-automat 19
2.3 Popis metody 21
2.4 Implementace metody 26
2.4.1 Robustní analyzátor 26
2.4.2 Postprefixový robustní analyzátor 31
2.5 Záruky metody 35
2.5.1 Význam značek ! a ? 36
2.5.2 Redukční jednoznačnost 40
2.5.3 Časová složitost 42
2.6 Shrnutí 44

3 Normalizace redukčních automatů **45**
3.1 Gramatika mon-red-automatu 45
3.1.1 Stromy výpočtů mon-red-automatu 45
3.1.2 Gramatika mon-red-automatu 49
3.2 Prefixově korektní mon-red-automat 62
3.3 Stavově minimální red-automat 66
3.4 Shrnutí 71

Závěr **73**

Seznam použité literatury **75**

Seznam tabulek **79**

Seznam obrázků **81**

Přílohy **83**

Úvod

Tato práce se zabývá *redukčními automaty*, jejich *normalizacemi* a využitím pro *robustní analýzu* a lokalizaci syntaktických chyb vzhledem k formálním jazykům ze třídy DCFL deterministických bezkontextových jazyků.

Redukční automat vychází z *automatu restartovacího*. Ten v roce 1995 zavedli Jančar, Mráz, Plátek a Vogel v článku [11] jako zařízení vhodné pro popis syntaxe jak formálních, tak i přirozených jazyků. Hlavním výsledkem zmíněného článku je charakterizace třídy DCFL pomocí deterministických monotónních restartovacích automatů.

Restartovací automat provádí *redukční analýzu* slova obsaženého v pracovním seznamu. Postupně ho zkracuje, přičemž zachovává jeho chybnost a (u deterministické verze automatu) i jeho bezchybnost. V původní definici je zkracování slova v seznamu uskutečňováno přepisováním nějakého jeho podslova omezené velikosti na podslovo kratší, vzniklé z původního vynecháním některých symbolů. Tento základní způsob zkracování je v pozdějších verzích restartovacího automatu rozšířen na přepisování omezeného podslova analyzovaného slova libovolným, kratším podslovem. V jednodušší verzi se používají pouze symboly vstupní abecedy, v obecnější verzi je dovoleno pracovat i se symboly pomocné abecedy. Každé zkrácení bezprostředně následuje vykonání *operace restart*. S touto operací přišli poprvé Kuboň s Plátkem v článku [13] roku 1994, tedy rok před představením restartovacího automatu, v rámci návrhu zařízení vhodného pro grammar-checking přirozených jazyků s volným slovosledem.

Teorie restartovacích automatů byla dále rozvíjena například v článcích [10], [12], [16], [17] nebo [18]. V článku [10] Jančar, Mráz, Plátek, Procházka a Vogel prověřují výpočetní sílu několika typů restartovacích automatů a porovnávají je s (deterministickými) bezkontextovými jazyky. Diskutují také vztah mezi restartovacími automaty a Marcusovými kontextuálními gramatikami a zavádějí třídu kontextuálních gramatik charakterizovanou *normálními* restartovacími automaty. Stejní autoři v článku [12] upouštějí od omezení velikosti podslova, ze kterého automat odstraňuje symboly. Třídu jazyků rozpoznávaných novým typem automatů, tzv. *vypouštějícími automaty* (angl. deleting automata), zasazují do Chomského hierarchie obohacené o třídy jazyků rozpoznávaných restartovacími automaty s různými omezeními (determinismus nebo monotonie). Porovnání vypouštějících automatů s Marcusovými gramatikami obsahuje článek [16] od Mráze, Plátka a Procházky. Podobnou problematikou se zabývá i [17] nebo [18].

Redukční automat jako modifikace restartovacího automatu se poprvé objevuje v [25]. Od restartovacího automatu se odlišuje

- explicitním určením odstraňovaných položek pracovního seznamu,
- přesunem obsahu výhledového okna do stavu řídící jednotky.

Explicitní určení odstraňovaných položek pracovního seznamu relativně vzhledem k poloze pracovní hlavy automatu je nezbytné pro přesné určení místa chyby v analyzovaném slově. Přesun obsahu výhledového okna do stavu řídící jednotky redukčního automatu umožňuje následující tři pohledy na redukční automat:

- redukční automat jako *konečný automat* rozpoznávající (regulární) jazyk přímo (bez redukcí) akceptovatelných slov,
- redukční automat jako *konečný převodník* definující binární relaci *redukce*,
- redukční automat jako *akceptor* modelující *redukční analýzu* i s vlastností *zachování chybnosti* a *bezchybnosti* postupně redukovaného řetězce.

Uvedené pohledy v této práci využíváme při přebírání technik a normalizací navržených v klasické teorii automatů a formálních jazyků a teorii překladu (minimalizace Moorova stroje, konstrukce položkového automatu k dané $\mathsf{LR}(0)$-gramatice, konstrukce prefixově korektního zásobníkového automatu).

Pro usnadnění orientace čtenáře v následujícím textu nyní stručně popíšeme obsah každé ze tří kapitol, ze kterých se tato práce skládá.

První kapitola definuje redukční automat a zavádí aparát pro popis jeho výpočtů a jejich členění do etap – částí výpočtu korespondujících s redukcemi slova v pracovním seznamu. Přes etapy výpočtu se tak dostáváme k redukční analýze. V jejím závěru ukazujeme různé způsoby reprezentace redukčních automatů.

Ve druhé kapitole navrhujeme metodu robustní redukční analýzy chybných slov pro libovolný deterministický jazyk zadaný *normovaným* monotónním redukčním automatem. Normovaným monotónním redukčním automatem chápeme libovolný monotónní redukční automat, který přijímá neprázdný jazyk, je prefixově korektní a stavově minimální. Konstrukci normovaného automatu k libovolnému monotónnímu redukčnímu automatu uvádíme ve třetí kapitole. Robustní analyzátor, kterým navrhovanou metodu implementujeme, přečte během analýzy celé vstupní slovo a lokalizuje v něm chyby určitých typů. Detekovanými typy chyb jsou prefixová a postprefixové nekorektnosti, které definujeme v téže kapitole. Během analýzy robustní analyzátor vkládá do vstupního slova pomocné symboly. Dva z těchto symbolů jsou značky, kterými analyzátor ve slově vymezuje zaručené (nezavlečené) chyby, zaručeně bezchybná podslova a místa redukčních konfliktů (podslova, která jsou v různých slovech redukována různým způsobem a mají tedy nejednoznačnou syntaktickou strukturu). Rovněž ukazujeme, že navrhovanou metodu lze implementovat deterministickým zásobníkovým převodníkem v lineárním čase vzhledem k délce vstupního slova.

Prefixové a postprefixové nekorektnosti vycházejí z neopravujícího zotavení ze syntaktických chyb, které roku 1985 publikoval Richter v článku [29]. Tento článek popisoval v obecné rovině vymezení maximálních korektních podslov v analyzovaném slově, ovšem bez konkrétní implementace pro obvyklé třídy formálních jazyků. Implementaci Richterovy metody pro speciální podtřídu deterministických

jazyků (generovaných gramatikami s (1-1) omezeným kontextem) publikoval Cormack roku 1989 v článku [3]. S jistým odstupem na Richterovu a Cormackovu práci navázali Deudekom s Grunem a Kooimanem (viz [6] a [7]), kteří Richterovu metodu zotavení z chyb použili pro řešení chybových stavů LLgen analyzátoru.

Richterův koncept vymezení maximálních *souvislých* korektních podslov resp. minimálních *souvislých* nekorektních podslov použil i autor této práce v definicích pojmů *korektního jádra* a *jednoduché syntaktické chyby* uvedených v jeho diplomové práci [23] z roku 1994. Z minimálních souvislých nekorektních podslov byla odvozena ještě kratší „nesouvislá“ nekorektní podslova odstraněním (z hlediska nekorektnosti „zbytečných“) symbolů. Tím se došlo ke konečně mnoha „základním“ syntaktickým chybám daného jazyka. Tyto typy chyb složené ze symbolů ležících na různých, od sebe vzdálených pozicích původního analyzovaného slova jsou významné pro přirozené jazyky s volným slovosledem, jakým je například čeština. Náš současný text ovšem rozvíjí tu část diplomové práce, která se zabývá „souvislými“ variantami chyb.

Třetí kapitola se zabývá normalizací monotónních redukčních automatů. Ukazuje, jak k libovolnému monotónnímu redukčnímu automatu sestrojit LR(0)-gramatiku generující právě jazyk rozpoznávaný daným automatem. Tuto gramatiku následně využíváme ke konstrukci ekvivalentního prefixově korektního monotónního redukčního automatu. Nakonec ještě předvádíme, jak minimalizovat množinu stavů libovolného redukčního automatu. Při minimalizaci stavů dochází i k odstranění všech nedosažitelných stavů. Jejich absence nám postačuje ke konstrukci postprefixového robustního analyzátoru, kterou popisujeme ve druhé kapitole.

Naším hlavním tématem je tedy redukční analýza a robustní redukční analýza. Tento lingvistický pohled přebíráme do jiné oblasti, a sice do oblasti syntaktické analýzy formálních jazyků zadaných normovanými monotónními redukčními automaty. Tyto automaty ovšem, na rozdíl od zařízení používaných v počítačové lingvistice, nepoužívají žádné kategorie ani pomocné symboly, pracují přímo na vstupnich slovech tvořených symboly konečné vstupní abecedy. Poznamenejme, že redukční analýza je jednou z lingvistických metod, kterou rozvíjí Jarmila Panevová se svými žáky. Příklad jejího použití nalezneme v [19].

Kapitola 1

Redukční automat

Redukční automat, stejně jako jeho předchůdce, *automat restartovací*, je zařízení vhodné pro popis syntaxe jak formálních, tak i přirozených jazyků. Vychází z představy *redukční analýzy*, tj. postupného zkracování analyzovaného řetězce, které zachovává jeho chybnost i jeho bezchybnost[1]. Jak je vidět na příkladu redukční analýzy aritmetického výrazu `a+((a)+a)`, je postupné zkracování vstupního řetězce zachovávající chybnost i bezchybnost poměrně přirozené.

```
a+((a)+a)
   ((a)+a)
   (  a  +a)
   (  a    )
      a
```

V této práci se budeme zabývat *monotónními* redukčními automaty. Ty totiž rozpoznávají právě *deterministické bezkontextové jazyky*, o které nám půjde především.

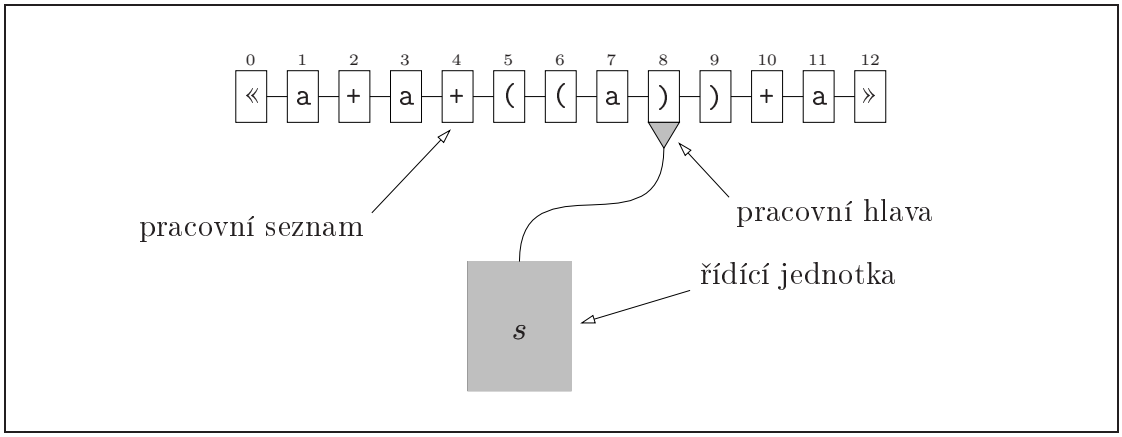
1.1 Definice a základní vlastnosti

Redukční automat si můžeme představit jako žáka, který provádí větný rozbor. Žák čte zadanou větu zleva doprava, jedno slovo po druhém, a aby se ve větě neztratil, ukazuje si v ní prstem. O přečtené části věty si dělá poznámky na kus papíru. Postupuje při tom takto:

1. Přesune svůj ukazovák nad další, dosud nepřečtené slovo.
2. Přečte si poznámku na kusu papíru a slovo, na které si ve větě ukazuje, a přepíše podle nich svůj poznámkový papír.
3. Podle nové poznámky na svém kusu papíru buď čte větu dále doprava, nebo větu prohlásí za bezchybnou nebo za chybnou, nebo větu zkrátí a s čistým kusem papíru ji začne číst znovu od začátku.

Větu zkrátí odstraněním několika slov pod a nalevo před ukazovákem. Vzdálenost všech odstraněných slov od ukazováku je přitom omezená nějakou konstantou.

[1] Restartovací automaty zachovávají bezchybnost, jsou-li deterministické. Redukční automaty jsou deterministické přímo z definice, jak uvidíme dále.



Obrázek 1.1: Redukční automat.

Redukční automat vidíme na obrázku 1.1. Automat zpracovává konečný *seznam položek*. První resp. poslední položka seznamu obsahuje *omezovač* « resp. », všechny ostatní položky obsahují symbol konečné *vstupní abecedy* (různý od obou omezovačů) a přirozené číslo, které nazýváme *pozicí* položky v seznamu. Pozice položek zleva doprava tvoří rostoucí posloupnost, pozice žádné položky se během výpočtu nemění. Redukční automat se stejně jako automat konečný skládá z *řídící jednotky* a *pracovní hlavy*. Řídící jednotka v sobě v každém okamžiku nese některý z konečně mnoha stavů automatu, pracovní hlava napojená na řídící jednotku snímá vždy jednu položku pracovního seznamu. Svou práci začíná automat v *počátečním stavu* s pracovní hlavou nad první položkou seznamu (obsahuje levý omezovač «). V každém kroku výpočtu automat přesune svou pracovní hlavu nad další položku vpravo a změní svůj stav podle současného stavu a podle symbolu snímané položky seznamu. Nový stav určí pomocí *přechodové funkce*.

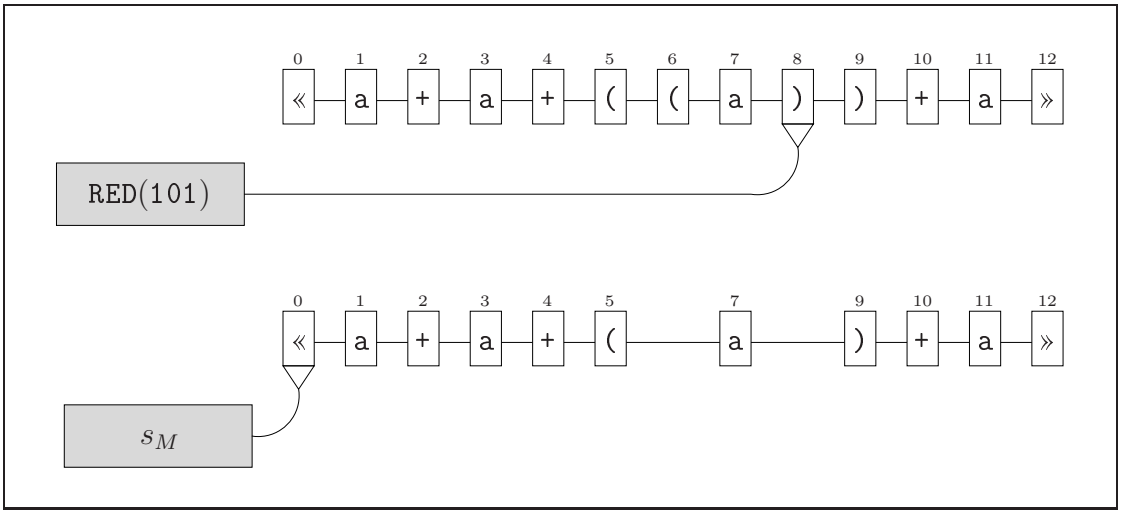
Některé stavy automatu jsou *koncové*. Red-automat obsahuje (nepovinně) koncové stavy `ACC` a `ERR` a konečně mnoho stavů z množiny $\{\texttt{RED}(n) \mid n \in \texttt{1}\cdot(\texttt{1}|\texttt{0})^*\}$. Koncové stavy ukončují výpočet automatu nebo určují, co má automat provést se seznamem položek. Jejich interpretace je následující:

`ACC` je *přijímající stav*. Přechod do tohoto stavu znamená ukončení výpočtu a přijetí slova obsaženého v pracovním seznamu.

`ERR` je *zamítající stav*. Přechodem do tohoto stavu automat ukončuje výpočet zamítnutím zpracovávaného slova.

$\texttt{RED}(n)$ je *redukční stav*. Po přechodu do tohoto stavu automat zkrátí pracovní seznam, přesune svou pracovní hlavu na jeho začátek a převede svou řídící jednotku do počátečního stavu s_M.

Parametr redukčního stavu, posloupnost nul a jedniček $n \in \texttt{1}\cdot(\texttt{1}|\texttt{0})^*$, budeme nazývat *redukční posloupností*. Redukční posloupnost určuje, které položky odstraní automat ze seznamu. Je-li zprava i-tý symbol redukční posloupnosti n roven `1`, pak automat odstraní i-tou položku seznamu počítáno od položky pod pracovní hlavou směrem doleva (položku pod pracovní hlavou přitom počítáme jako první). Příklad zkrácení slova « `a+a+((a))+a` » podle redukční posloupnosti `101` vidíme na obrázku 1.2. Koncové stavy vždy končí určitou *etapu* výpočtu automatu. Etapou myslíme libovolnou část výpočtu, kterou automat začíná v počátečním



Obrázek 1.2: Zkrácení pracovního seznamu.

stavu nad první položkou seznamu, a která končí, jakmile se automat dostane do libovolného koncového stavu.

Definice redukčního automatu. Formálně definujeme redukční automat M jako sedmici

$$M = (\Sigma_M, «, », S_M, s_M, F_M, f_M),$$

kde

Σ_M je konečná *vstupní abeceda*,

$«,» \notin \Sigma_M$ je *levý* resp. *pravý omezovač* vstupního slova,

S_M je konečná množina *nekoncových stavů*,

$s_M \in S_M$ je *počáteční stav*,

F_M je konečná množina *koncových stavů* neboli *operací* disjunktní s množinou S_M nekoncových stavů, obsahující (nepovinně) přijímající koncový stav `ACC` a zamítající koncový stav `ERR` a konečně mnoho redukčních koncových stavů z množiny $\{\mathtt{RED}(n) \mid n \in \mathtt{1}\cdot(\mathtt{1}|\mathtt{0})^*\}$,

$f_M : S_M \times (\Sigma_M \cup \{»\}) \longrightarrow (S_M \cup F_M)$ je *přechodová funkce*, která libovolné dvojici tvořené nekoncovým stavem a symbolem vstupní abecedy nebo pravým omezovačem » přiřadí stav (nekoncový nebo koncový) a splňuje následující podmínky: Libovolné dvojici tvořené nekoncovým stavem a pravým omezovačem » přiřadí vždy koncový stav,

$$f_M(s, ») = s' \implies s' \in F_M.$$

Je-li navíc tento koncový stav redukční, pak jeho redukční posloupnost končí symbolem `0`,

$$f_M(s, ») = \mathtt{RED}(n) \implies \mathtt{0} \text{ je sufix } n.$$

Je-li hodnotou přechodová funkce přijímající koncový stav ACC, pak je druhým argumentem přechodové funkce pravý omezovač »,

$$f_M(s,a) = \texttt{ACC} \implies a = \text{»}.$$

Pokud $f_M(s,a) = s'$, říkáme, že redukční automat M *přejde* ze stavu s přes symbol a do nového stavu s', nebo také, že přechodová funkce *převede řídící jednotku* redukčního automatu M ze stavu s přes symbol a do nového stavu s'.

Rozšíření přechodové funkce. Chování redukčního automatu budeme popisovat pomocí dvou funkcí rozšiřujících přechodovou funkci f_M. Rozšiřující funkce poslouží jednak pro zavedení relace redukce a později pro formulaci podmínek a operací robustní redukční analýzy.

$$\delta_M : (S_M \cup F_M \cup \{\texttt{RED}\}) \times (\Sigma_M \cup \{\text{»}\}) \longrightarrow (S_M \cup F_M \cup \{\texttt{RED}\})$$
$$\Delta_M : (S_M \cup F_M^*) \times (\Sigma_M \cup \{\text{»}\}) \longrightarrow (S_M \cup F_M^*)$$

RED je nový pomocný stav různý od všech stavů z S_M a množinu F_M^* definujeme takto:

$$F_M^* = F_M \cup \{\texttt{RED}(n \cdot \texttt{0}^k) \mid \texttt{RED}(n) \in F_M \text{ a } k \geq 1\}$$

Povšimněme si, že F_M^* není konečná.

Obě funkce δ_M, Δ_M se pro všechny dvojice tvořené stavem $s \in S_M$ a symbolem $a \in (\Sigma_M \cup \{\text{»}\}$ shodují s přechodovou funkcí f_M. Pro všechny ostatní dvojice z příslušných definičních oborů definujeme nové funkce takto:

$$\begin{aligned} \delta_M(\texttt{ACC}, a) &= \texttt{ACC} & \Delta_M(\texttt{ACC}, a) &= \texttt{ACC} \\ \delta_M(\texttt{ERR}, a) &= \texttt{ERR} & \Delta_M(\texttt{ERR}, a) &= \texttt{ERR} \\ \delta_M(\texttt{RED}(n), a) = \delta_M(\texttt{RED}, a) &= \texttt{RED} & \Delta_M(\texttt{RED}(n), a) &= \texttt{RED}(n \cdot \texttt{0}) \end{aligned}$$

Obvyklým způsobem zavedeme funkci δ_M^* resp. Δ_M^* jako reflexivní a tranzitivní uzávěr funkce δ_M resp. Δ_M. Obě funkce definujeme i pro konečné podmnožiny S množiny $S_M \cup F_M \cup \{\texttt{RED}\}$ resp. $S_M \cup F_M^*$,

$$\delta_M^*(S,u) = \{\delta_M^*(s,u) \mid s \in S\}, \qquad \Delta_M^*(S,u) = \{\Delta_M^*(s,u) \mid s \in S\}.$$

Charakteristická konstanta. Pro automat M zavedeme tzv. *charakteristickou konstantu* jako délku nejdelší binární posloupnosti obsažené v nějakém redukčním stavu automatu M,

$$k_M = \max\big\{|n| \mid \texttt{RED}(n) \in F_M\big\}.$$

Operace a relace redukce. Zkrácení slova podle redukční posloupnosti budeme zapisovat pomocí operace redukce / takto:

$$\begin{aligned} a/\texttt{0} &= a & \lambda/n &= \lambda & (u \cdot a)/\lambda &= u \cdot a \\ a/\texttt{1} &= \lambda & u/\lambda &= u & (u \cdot a)/(n \cdot i) &= (u/n) \cdot (a/i) \end{aligned}$$

kde $u \in \Sigma_M^*$, $a \in \Sigma_M \cup \{»\}$, $n \in (\mathtt{10}^*)^*$ a $i \in \{\mathtt{0}, \mathtt{1}\}$. Slovo u ani redukční posloupnost n zde neomezujeme žádnou konstantou, u může být delší než n i naopak. Zkrácení slova `(a)` resp. `+a»` podle redukční posloupnosti `101` resp. `110` pak vypadá takto:

$$\begin{array}{cccc} & (& \mathtt{a} &) \\ / & 1 & 0 & 1 \\ \hline = & & \mathtt{a} & \end{array} \qquad\qquad \begin{array}{cccc} & + & \mathtt{a} & » \\ / & 1 & 1 & 0 \\ \hline = & & & » \end{array}$$

Pomocí operace redukce zavedeme *relaci redukce*. Ta umožňuje popsat, jak automat postupně zkracuje zpracovávané slovo $w \in \Sigma_M^*$. Redukční automat M *redukuje* slovo «w» na slovo «w'»,

$$«w» \Rightarrow_M «w'»,$$

jestliže $\Delta_M^*(s_M, w») = \mathtt{RED}(n)$ a «w»$/n =$ «w'». Zřejmě je $|w| > |w'|$ a uvedená redukce je *zkracující*. Reflexivní a tranzitivní uzávěr relace redukce budeme označovat jako $\Rightarrow_M^*$.

Redukční analýza. *Redukční analýzou* automatu M myslíme libovolnou posloupnost redukcí

$$«w_1» \Rightarrow «w_2» \Rightarrow \ldots \Rightarrow «w_n»,$$

kterou nelze dále prodloužit. Je-li $\delta_M^*(w_n») = \mathtt{ACC}$, mluvíme o *přijímající* redukční analýze, jinak se jedná o *zamítající* redukční analýzu. Místo termínu redukční analýza budeme často používat zkrácený termín analýza.

Přijímaný jazyk. Slova nad abecedou Σ_M přijatá redukčním automatem M v jediné etapě tvoří jeho *jednoduchý jazyk*

$$L_0(M) = \big\{w \in \Sigma_M^* \mid \delta_M^*(s_M, w») = \mathtt{ACC}\big\}.$$

Není těžké nahlédnout, že $L_0(M)$ je regulární jazyk.

Jazyk *přijímaný* nebo též *rozpoznávaný* redukčním automatem M definujeme jako množinu právě všech slov nad abecedou Σ_M, pro která existuje přijímající analýza automatu M,

$$L(M) = \big\{w \in \Sigma_M^* \mid «w» \Rightarrow_M^* «w'» \text{ a } w' \in L_0(M)\big\}.$$

Ekvivalence red-automatů. Pomocí rovnosti množin přijímaných jazyků definujeme ekvivalenci na třídě všech redukčních automatů. Dva automaty M_1 a M_2 jsou *ekvivalentní*, jestliže

$$L(M_1) = L(M_2).$$

Platí-li pro red-automaty M_1, M_2 zároveň

$$L_0(M_1) = L_0(M_2) \quad \text{a} \quad «w» \Rightarrow_{M_1} «w'» \iff «w» \Rightarrow_{M_2} «w'»$$

pro libovolná w, $w' \in (\Sigma_{M_1}^* \cup \Sigma_{M_2}^*)$, pak jsou tyto automaty zřejmě ekvivalentní a říkáme, že jsou *redukčně ekvivalentní*. Redukční ekvivalenci automatů můžeme

ještě zesílit tím, že budeme navíc požadovat, aby oba automaty redukovaly resp. přijaly resp. zamítly seznam obsahující stejné slovo na stejném místě. V takovém případě je označíme za *silně ekvivalentní*. Formálně, redukční automaty M_1 a M_2 jsou silně ekvivalentní, jestliže pro libovolný prefix «u slova «w» obsaženého v seznamu platí zároveň

$$\begin{aligned} \delta^*_{M_1}(s_{M_1}, u) = \texttt{RED}(n) &\iff \delta^*_{M_2}(s_{M_2}, u) = \texttt{RED}(n) \\ \delta^*_{M_1}(s_{M_1}, u) = \texttt{ACC} &\iff \delta^*_{M_2}(s_{M_2}, u) = \texttt{ACC} \\ \delta^*_{M_1}(s_{M_1}, u) = \texttt{ERR} &\iff \delta^*_{M_2}(s_{M_2}, u) = \texttt{ERR} \end{aligned}$$

Úmluva. Dále budeme uvažovat jen redukční automaty neredukující před začátkem zpracovávaného slova, tedy jen redukční automaty splňující pro libovolné slovo $u \in (\Sigma_M \cup \{»\})^*$ a redukční posloupnost n následující podmínku:

$$\delta^*_M(s_M, u) = \texttt{RED}(n) \implies |u| \geq |n|$$

K libovolnému redukčnímu automatu M zřejmě snadno sestrojíme ekvivalentní redukční automat M' vyhovující uvedené úmluvě. Stačí k tomu do řídící jednotky přidat počítadlo vzdálenosti pracovní hlavy od levého omezovače «. Počátečním stavem nového automatu M' bude dvojice $\langle s_M, 1\rangle$ tvořená počátečním stavem původního automatu M a počítadlem nastaveným na 1. Přechodová funkce automatu M' zohlední počítadlo v řídící jednotce takto: V každém z prvních $k_M - 1$ přesunů pracovní hlavy zvýší automat M' počítadlo o 1, v následujících přesunech ponechává počítadlo beze změny. Při přechodu do redukčního koncového stavu ze stavu s počítadlem nastaveným na k automat M' redukuje podle sufixu redukční posloupnosti n délky k kromě případu, kdy $k = 1$ a automat snímá pravý omezovač » (pracovní seznam obsahuje jen omezovače a není tedy již co redukovat). V takovém případu automat přejde do zamítajícího koncového stavu `ERR`. Přechod do přijímajícího stavu `ACC` resp. zamítajícího koncového stavu `ERR` není počítadlem ovlivněno.

Vlastnost zachování chybnosti a bezchybnosti. Pomocí relace redukce můžeme vyslovit základní vlastnost, kterou splňují všechny redukční automaty:

Lemma 1.1.1. *Jestliže «w_1» $\Rightarrow_M$ «w_2», potom $w_1 \in L(M)$, právě když $w_2 \in L(M)$.*

Důkaz. Platnost lemmatu snadno nahlédneme z následující úvahy: Je-li «w_2» $\Rightarrow^*_M$ «w» přijímající analýza pro slovo w_2, pak «w_1» $\Rightarrow_M$ «w_2» $\Rightarrow^*_M$ «w» je přijímající analýza pro slovo w_1. Vyjdeme-li naopak z předpokladu, že $w_1 \in L(M)$, pak máme analýzu «w_1» $\Rightarrow^*_M$ «w», kde w je nějaké slovo z $L(M)$. Tato analýza musí začínat redukcí slova w_1 na slovo w_2, jinak by nebyla přechodová funkce automatu M definována jednoznačně. □

Tuto vlastnost budeme nazývat *vlastností zachovávání chybnosti a bezchybnosti.*

Monotonie. Podobně jako v [11] pro restartovací automaty zavedeme *monotonii* i pro red-automaty. Red-automat je *monotonní*, jestliže v libovolné etapě navštíví všechny položky, které navštívil v předchozí etapě a které zůstaly v pracovním seznamu. To samé můžeme vyjádřit i tak, že automat ukončí následující etapu přechodem do nějakého koncového stavu nejdříve po přesunu přes slovo w' tvořené všemi symboly navštívenými v předchozí etapě, které zůstaly v pracovním seznamu, a tedy, že se po přesunu přes slovo $w'/\mathtt{1}$ (tvořené slovem w' bez posledního symbolu vpravo) dostane do nekoncového stavu. Formálně definujeme monotonii redukčního automatu takto: Nechť M je red-automat, s_M je jeho počáteční stav a δ_M je jeho rozšířená přechodová funkce. M je monotonní, pokud je splněna následující podmínka:

$$\forall w, n : \delta_M^*(s_M, w) = \mathtt{RED}(n) \implies \delta_M^*(s_M, (w/n)/\mathtt{1}) \in S_M$$

V práci několikrát zmiňujeme ostrou monotonii. Red-automat je *ostře monotonní*, jestliže v libovolné etapě navštíví alespoň jednu položku poprvé, formálně:

$$\forall w, n : \delta_M^*(s_M, w) = \mathtt{RED}(n) \implies \delta_M^*(s_M, w/n) \in S_M$$

Monotonie je důležitá pro charakterizaci třídy DCFL pomocí podtřídy restartovacích automatů. Deterministické monotonní restartovací automaty (det-mon-R-automaty) rozpoznávají právě všechny deterministické bezkontextové jazyky. Stejným způsobem můžeme třídu DCFL charakterizovat i pomocí monotónních redukčních automatů. Restartovací automat se od redukčního automatu liší jen tím, že jeho pracovní hlava je rozšířena o tzv. výhledové okno, kterým snímá souvislé podslovo pevné délky napravo od pracovní hlavy, a že redukuje symboly snímané pracovní hlavou a výhledovým oknem. Libovolný restartovací automat můžeme tedy simulovat redukčním automatem, který si obsah výhledového okna uchovává ve své řídící jednotce. Naopak, každý redukční automat můžeme simulovat restartovacím automatem s výhledovým oknem velikosti k, kde k je maximální délka redukční posloupnosti redukčního automatu. Přitom zřejmě platí, že v každé etapě navštíví simulovaný i simulující automat stejné položky pracovního seznamu, takže monotonie je zachována.

1.2 Reprezentace red-automatů

Lepší představu o daném redukčním automatu dává jeho vhodná reprezentace. Zde předvedeme následující tři způsoby reprezentace:

- přechodová tabulka,
- přechodový graf,
- redukční a akceptační pravidla.

První dva způsoby reprezentace jsou převzaty z teorie konečných automatů, poslední z přepisovacích systémů. Dále budeme používat první z nich, tedy reprezentaci přechodovou tabulkou.

δ_M	a	+	(	)	»
⇒ s_0	s_1	ERR	s_4	ERR	ERR
s_1	ERR	s_2	ERR	ERR	ACC
s_2	s_3	ERR	s_4	ERR	ERR
s_3	ERR	s_2	ERR	RED(110)	RED(110)
s_4	s_5	ERR	s_4	ERR	ERR
s_5	ERR	s_2	ERR	RED(101)	ERR

Tabulka 1.1: Přechodová tabulka.

Všechny tři uvedené způsoby reprezentace si nyní ukažme na příkladu redukčního automatu M, který rozpoznává jazyk zjednodušených aritmetických výrazů generovaný gramatikou s pravidly

$$S \to \mathtt{a} \mid S\mathtt{+}S \mid \mathtt{(}S\mathtt{)}$$

Formálně je automat definován jako sedmice $(\Sigma_M, «, », S_M, s_M, F_M, \delta_M)$, kde

$$\Sigma_M = \{\, \mathtt{a}, \mathtt{+}, \mathtt{(}, \mathtt{)} \,\}$$

$$S_M = \{\, s_0, s_1, s_2, s_3, s_4, s_5 \,\}$$

$$F_M = \{\, \mathtt{RED(110)}, \mathtt{RED(101)}, \mathtt{ACC}, \mathtt{ERR} \,\}$$

$$s_M = s_0$$

$$\delta_M = \{$$

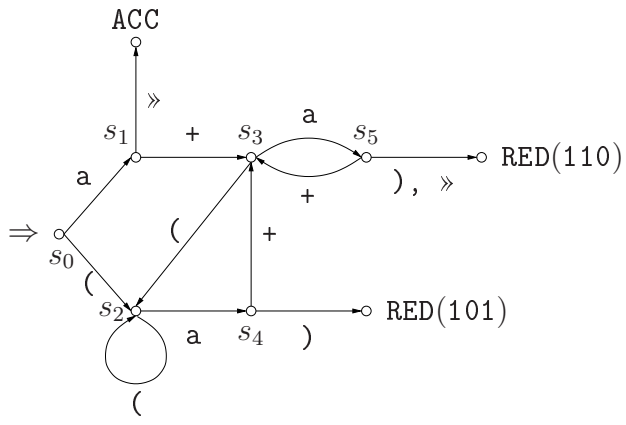
$$\begin{array}{lllll}
(s_0, \mathtt{a}, s_1\ \ ), & (s_0, \mathtt{+}, \mathtt{ERR}), & (s_0, \mathtt{(}, s_4\ \ ), & (s_0, \mathtt{)}, \mathtt{ERR}\ \ \ \ \ ), & (s_0, », \mathtt{ERR}\ \ \ \ \ ), \\
(s_1, \mathtt{a}, \mathtt{ERR}), & (s_1, \mathtt{+}, s_2\ \ ), & (s_1, \mathtt{(}, \mathtt{ERR}), & (s_1, \mathtt{)}, \mathtt{ERR}\ \ \ \ \ ), & (s_1, », \mathtt{ACC}\ \ \ \ \ ), \\
(s_2, \mathtt{a}, s_3\ \ ), & (s_2, \mathtt{+}, \mathtt{ERR}), & (s_2, \mathtt{(}, s_4\ \ ), & (s_2, \mathtt{)}, \mathtt{ERR}\ \ \ \ \ ), & (s_2, », \mathtt{ERR}\ \ \ \ \ ), \\
(s_3, \mathtt{a}, \mathtt{ERR}), & (s_3, \mathtt{+}, s_2\ \ ), & (s_3, \mathtt{(}, \mathtt{ERR}), & (s_3, \mathtt{)}, \mathtt{RED(110)}), & (s_3, », \mathtt{RED(110)}), \\
(s_4, \mathtt{a}, s_5\ \ ), & (s_4, \mathtt{+}, \mathtt{ERR}), & (s_4, \mathtt{(}, s_4\ \ ), & (s_4, \mathtt{)}, \mathtt{ERR}\ \ \ \ \ ), & (s_4, », \mathtt{ERR}\ \ \ \ \ ), \\
(s_5, \mathtt{a}, \mathtt{ERR}), & (s_5, \mathtt{+}, s_2\ \ ), & (s_5, \mathtt{(}, \mathtt{ERR}), & (s_5, \mathtt{)}, \mathtt{RED(101)}), & (s_5, », \mathtt{ERR}\ \ \ \ \ )
\end{array}$$

$$\}$$

Tentýž automat můžeme přehledněji popsat tabulkou 1.1, kterou čteme takto: Je-li v políčku na řádku označeném stavem s a ve sloupci označeném symbolem a stav s', pak platí $\delta_M(s, a) = s'$ a naopak. Řádek obsahující přechody s počátečním vstupním stavem je v tabulce označen šipkou (⇒).

Druhým způsobem reprezentace redukčního automatu je přechodový graf (viz obrázek 1.3). Jeho vrcholy jsou stavy nebo operace daného automatu, hrany jsou ohodnoceny symboly vstupní abecedy nebo pravým omezovačem ». Počáteční stav je v grafu, podobně jako v tabulce, označen šipkou (⇒). Je-li a ohodnocením hrany vedoucí z vrcholu s do vrcholu s', pak pro příslušný automat platí $\delta_M(s, a) = s'$. Aby byl graf přehlednější, nekreslíme do něho přechody do stavu ERR a více přechodů, které se shodují vstupními stavy a výstupními stavy nebo operacemi znázorňujeme jedinou hranou ohodnocenou více symboly.

Reprezentaci red-automatu přepisovacími pravidly odvodíme z tabulky jeho přechodové funkce. Každý stav automatu jednoznačně koresponduje s množinou slov, která do něho převedou automat ze stavu počátečního (ten budeme reprezentovat levým omezovačem «). Množina takových slov je zřejmě regulárním



Obrázek 1.3: Přechodový graf.

	a	+	(	)	»
⇒ $s_0 = «$	s_1	ERR	s_4	ERR	ERR
$s_1 = «\,\mathtt{a}$	ERR	s_2	ERR	ERR	ACC
$s_2 = «\,(\mathtt{a}+)^+$	s_3	ERR	s_4	ERR	ERR
$s_3 = «\,((^*\,\mathtt{a}+)^+\,\mathtt{a}$	ERR	s_2	ERR	RED(110)	RED(110)
$s_4 = «\,((\mathtt{a}+)^*\,()^+$	s_5	ERR	s_4	ERR	ERR
$s_5 = «\,((\mathtt{a}+)^*\,()^+\,\mathtt{a}$	ERR	s_2	ERR	RED(101)	ERR

Tabulka 1.2: Stavy jako regulární výrazy.

jazykem, takže stav můžeme zapsat regulárním výrazem, který odpovídá tomuto jazyku. Mezistupeň mezi přechodovou tabulkou a redukčnímu/akceptačními pravidly uvádíme v tabulce 1.2.

Když odhlédneme od „nedůležitých“ stavů na cestě od počátečního stavu k operaci ukončující etapu práce automatu, získáme množinu redukčních (resp. akceptačních) pravidel uvedenou na obrázku 1.4.

1.3 Shrnutí

V této kapitole jsme definovali redukční automat a zavedli jsme aparát pro popis jeho výpočtů a jejich členění do etap – částí výpočtu korespondujících s redukcemi slova v pracovním seznamu. Etapy výpočtu nás dovedly k redukční analýze. Od restartovacích automatů jsme převzali monotonii a ostrou monotonii. Monotonie je důležitá vlastnost, která umožňuje charakterizovat třídu DCFL deterministických bezkontextových jazyků pomocí restartovacích (resp. redukčních) automatů. Ukázali jsme také různé způsoby reprezentace redukčních automatů – přechodo-

$$
\begin{aligned}
«\,\mathtt{a}\,» &\rightarrow \mathtt{ACC}\\
«\,((^*\,\mathtt{a}+)^+\,\mathtt{a}\,()|») &\rightarrow \mathtt{RED(110)}\\
«\,((\mathtt{a}+)^*\,()^+\,\mathtt{a}\,) &\rightarrow \mathtt{RED(101)}
\end{aligned}
$$

Obrázek 1.4: Redukční a akceptační pravidla.

vou tabulkou, přechodovým grafem a redukčními a akceptačními pravidly.

Kapitola 2

Redukční analýza chybných slov

Na rozdíl od překladačů, kde je k dispozici jak rozsáhlá teorie, tak i řada praktických nástrojů (např. `yacc`), které z konstrukce překladače programovacího jazyka dělají rutinní záležitost, je zotavení z chyb při syntaktické analýze stále spíše otázkou šikovnosti a umu programátora bez jasných formálních záruk.

Pojem „zotavení“ (recovery) předjímá způsob řešení situace, kdy překladač narazí na chybu. Takovým řešením bývá obnovení práce překladače různými prostředky: změnou stavu řídící jednotky, přeskočením části vstupu pracovní hlavou, přepsáním části zásobníku nebo opravou části vstupu. O co nám ale ve skutečnosti jde, je určení (lokalizace, detekce) pokud možno všech „skutečných“ chyb ve vstupním slově. Zotavení není cílem, ale jednou z metod, jak skutečného cíle dosáhnout.

Chyba se obvykle definuje jako nemožnost překladače pokračovat v překladu. Uživateli se oznamuje hláškou typu

> „Neočekávaný symbol a na pozici p.“

Taková definice chyby spolu s tím, že zotavení se ne vždy povede, má za důsledek takzvané „zavlečené“ chyby. Těm se (na rozdíl od chyb „skutečných“) snažíme vyhnout.

My postupujeme jinak. Nejprve říkáme, co v analyzovaných slovech hledáme, a pak navrhujeme, jak to hledat. Hledanými chybami jsou zde prefixové a postprefixové nekorektnosti. Tyto pojmy přejímáme s pozměněnými jmény z [29, 3]. Definujeme je pomocí maximálních (souvislých) podslov analyzovaného slova, která jsou obsažena v nějakém slově daného jazyka. Tento přístup je univerzálně použitelný pro všechny třídy jazyků. Následně, v části 2.3, popisujeme metodu lokalizace těchto prefixových a postprefixových nekorektností použitelnou pro libovolný normovaný mon-red-automat rozpoznávající neprázdný jazyk. Ten představujeme v části 2.2. Navrženou metodu implementujeme robustním analyzátorem, novým typem zařízení, které zavádíme v části 2.4 spolu s postupem, jak ho pro libovolný normovaný mon-red-automat zkonstruovat.

Jsme přesvědčeni, že se redukční automat ukáže jako zařízení, které umožňuje poměrně snadné a srozumitelné vyjadřování v oblasti syntaktických nekorektností a jejich lokalizace.

2.1 Prefixové a postprefixové (ne)korektnosti[1]

Předpokládejme, že L je neprázdný jazyk nad abecedou Σ, «, » $\notin \Sigma$, a že každý symbol této abecedy se vyskytuje v nějakém slově jazyka L. Definice prefixové a postprefixových (ne)korektností ve slovech z množiny «Σ^*» vzhledem k jazyku L je nezávislá na způsobu přijímání resp. generování slov tohoto jazyka. Vychází pouze z vymezení jazyka L jako podmnožiny množiny Σ^* a z množiny všech jeho podslov. Dané vstupní slovo může obsahovat podslova, která sama o sobě mohou být podslovy nějakých slov daného jazyka, ale dohromady k sobě nepatří. Žádnému takovému správnému podslovu nepřisuzujeme větší význam než kterémukoli jinému. Analyzátor nemůže rozhodnout, které z daných podslov je skutečně správné a které ne; nemůže vědět, jak analyzované slovo opravit na slovo, které měl na mysli jeho autor. Proto analýzu stavíme na vymezení takových podslov zadaného slova, která jsou podslovem nějakého (jiného) slova jazyka L nebo naopak nejsou podslovem žádného slova tohoto jazyka. Kromě toho se snažíme zcela se vyhnout lokalizaci falešných (nesprávných) chyb.

Slovo v tvořené symboly abecedy Σ případně začínající levým omezovačem « nebo končící pravým omezovačem » nazveme *nekorektním* vzhledem k jazyku L, jestliže $uvw \notin$ «L» pro libovolná u, w taková, že $uvw \in$ «Σ^*». Takto definované nekorektní slovo může obsahovat vlastní podslova, která jsou sama nekorektní. Abychom se dostali k „podstatě“ nekorektnosti, definujeme nekorektní jádro: Slovo v nazveme *nekorektním jádrem* slova w vzhledem k jazyku L, jestliže je nekorektní vzhledem k jazyku L a je minimální v uspořádání „býti podslovem“. Na druhou stranu, slovo v je *korektní* podslovo slova $w \in$ «Σ^*» vzhledem k jazyku L, jestliže $w = xvy$ a pro nějaká x', y' je $x'vy' \in$ «L». Slovo v je *korektním jádrem* ve slově w, je-li korektním podslovem slova w vzhledem k L a je maximální v uspořádání „býti podslovem“. Předpoklad, že každý symbol abecedy Σ se vyskytuje v nějakém slově jazyka L, nám zaručuje, že každý symbol libovolného slova $w \in$ «Σ^*» je obsažen v nějakém korektním jádru tohoto slova.

Tvrzení 2.1.1. *Je-li v korektní jádro v $uavw$, resp. $uvbw$ vzhledem k L, pak av, resp. vb je nekorektním podslovem slova $uavw$, resp. $uvbw$ vzhledem k L.*

Tvrzení 2.1.2. *Jsou-li vw, wx dvě různá korektní jádra v $uvwxy$ vzhledem k L, pak vwx je nekorektním podslovem slova $uvwxy$.*

Důkaz. Kdyby existovala u_1, y_1 taková, že $u_1vwxy_1 \in$ «L», pak by vwx bylo korektním podslovem a vw nebo wx by šlo prodloužit na větší korektní podslovo. Aspoň jedno z podslov vw, wx by tedy nebylo maximálním korektním podslovem. □

Tvrzení 2.1.3. *Je-li avb nekorektní jádro vzhledem k L, pak av a vb jsou jediná korektní jádra v avb vzhledem k L.*

Důkaz. Z našeho předpokladu zřejmě plyne, že každý symbol slova $w = avb$ je částí nějakého korektního jádra ve w.

Kdyby w obsahovalo jediné korektní jádro, pak by tímto korektním jádrem bylo celé w a tudíž by w nebylo nekorektní.

[1] Tato podkapitola je přejata jen s malými změnami z autorovy diplomové práce.

Kdyby w obsahovalo více než dvě korektní jádra, pak by podle tvrzení 2.1.2 bylo slovo tvořené dvěma sousedními korektními jádry bylo nekorektní a tudíž by w nebylo nekorektním jádrem. □

Tvrzení 2.1.4. *Pokud jsou uav a vbw jediná korektní jádra v uavbw vzhledem k L, pak avb je nekorektní jádro v uavbw vzhledem k L.*

Důkaz. Kdyby $z = avb$ bylo korektní, pak by nastala právě jedna z nasledujících možností:

(a) uav nebo vbw není korektní jádro v $uavbw$,

(b) existuje třetí korektní jádro v $uavbw$.

Podslovo avb je minimální, protože av i vb jsou korektní podslova. □

Na základě předchozích definic a tvrzení můžeme v libovolném slově «w» $\in$ «Σ^*» vymezit korektní a nekorektní jádra vzhledem k libovolnému jazyku L nad abecedou Σ.

Nekorektní jádra můžeme popsat následujícím způsobem:

nekorektní jádro na pozicích	p_1	až	p_3
nekorektní jádro na pozicích	p_2	až	p_4
nekorektní jádro na pozicích	p_5	až	p_6

Prefixová korektnost je zleva první korektní jádro analyzovaného slova. Protože analyzované slovo vždy začíná levým omezovačem «, je zleva první korektní jádro prefixem nějakého slova z «L». *Postprefixové korektnosti* jsou sufixy korektních jader vpravo za prefixovou resp. předchozí postprefixovou korektností. Právě zavedené pojmy jsou znázorněné na obrázku 2.1.

Zde nám půjde o deterministické a co nejpřesnější vyhledání posloupností (nepřekrývajících se) postprefixových korektností v analyzovaných slovech.

2.2 Normovaný mon-red-automat

Robustní analyzátor budeme konstruovat pro speciální podtřídu monotónních redukčních automatů, které budeme nazývat *normované* mon-red-automaty. Normovaný je libovolný mon-red-automat M, který je současně

- *prefixově korektní*,
- *stavově minimální*.

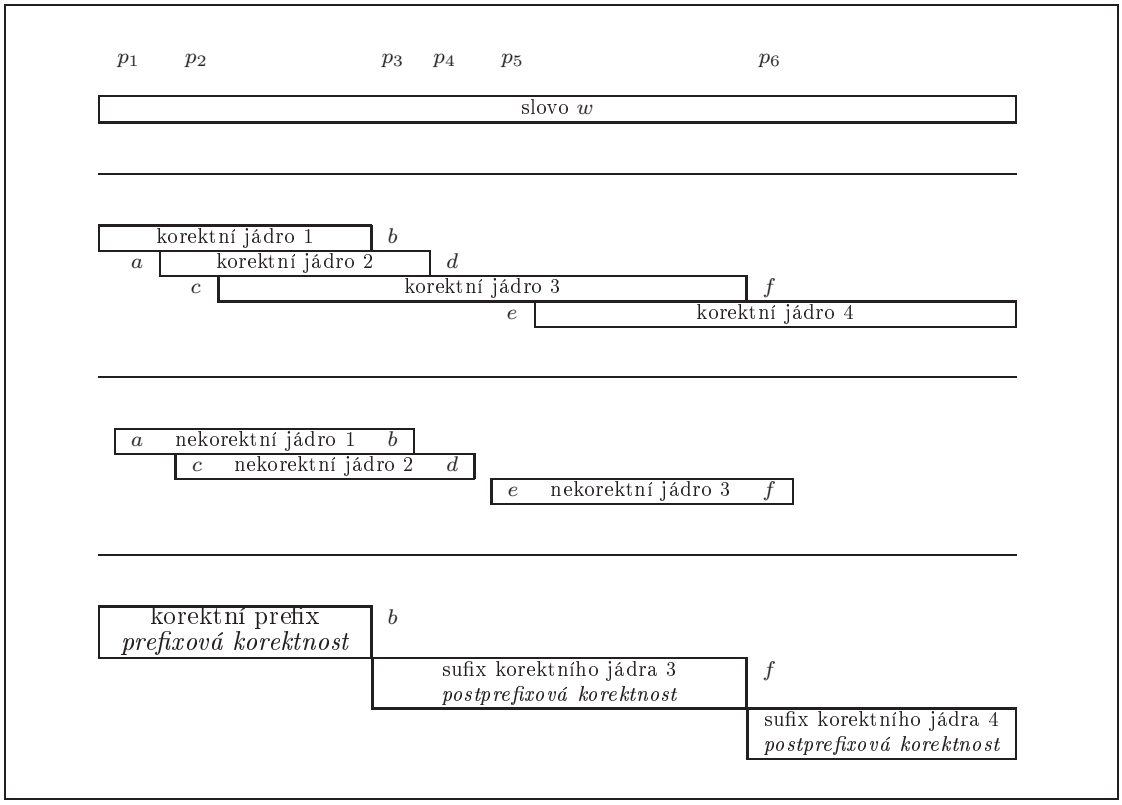
Prefixově korektní je redukční automat M, platí-li pro libovolné slovo $u \in \Sigma_M^*$, a redukční posloupnost n následující implikace:

$$\delta_M^*(s_M, u) \in S_M \implies \exists v \in \Sigma_M^* : uv \in L(M) \tag{2.1}$$

$$\delta_M^*(s_M, u) = \texttt{RED}(n) \implies \exists v \in \Sigma_M^* : uv \in L(M) \tag{2.2}$$

$$\delta_M^*(s_M, u») = \texttt{RED}(n) \implies u \in L(M) \tag{2.3}$$

První dvě implikace (2.1) a (2.2) říkají, že libovolné slovo u nad abecedou Σ_M, přes které automat M přesune svou pracovní hlavu a přitom převede svou řídící



Obrázek 2.1: (Ne)korektní jádra a (post)prefixové korektnosti.

jednotku z počátečního stavu s_M do libovolného nekoncového nebo redukčního stavu, je prefixem nějakého slova jazyka $L(M)$.

Třetí implikace (2.3) tvrdí, že libovolné slovo u nad abecedou Σ_M, přes které automat M přesune svou pracovní hlavu z počátečního stavu s_M, za ním ještě překročí pravý omezovač » a přitom převede svou řídící jednotku do libovolného redukčního stavu, je již jistě slovem jazyka $L(M)$. Přitom k formálnímu přijetí slova u může automat potřebovat ještě několik dalších etap.

Stavově minimální je libovolný redukční automat M, který má dosažitelné všechny (nekoncové i koncové) stavy,

$$\forall s \in S_M \; \exists u \in \Sigma_M^* : \delta_M^*(s_M, u) = s, \tag{2.4}$$

a libovolné dva jeho *etapově ekvivalentní* nekoncové stavy jsou totožné. Nekoncové stavy s_1, $s_2 \in S_M$ automatu M jsou *etapově ekvivalentní*, platí-li pro libovolné $u \in (\Sigma_M)^* \cdot \{»\}$ následující implikace:

$$\big(\; \delta_M^*(s_1, u) \in F_M \;\text{ nebo }\; \delta_M^*(s_2, u) \in F_M \; \big) \implies \delta_M^*(s_1, u) = \delta_M^*(s_2, u)$$

Pro metodu lokalizace postprefixových (ne)korektností stačí pouze dosažitelnost všech stavů redukčního automatu. Minimalita počtu (nekoncových) stavů je ale pro naše následující záměry výhodná.

Pro tuto kapitolu nám postačují definice uvedených pojmů. Podrobněji se jimi zabýváme v kapitole 3, kde uvádíme i konstrukce ekvivalentních automatů s těmito vlastnostmi. Důležité je, že pro libovolný mon-red-automat M přijímající

neprázdný jazyk umíme zkonstruovat ekvivalentní normovaný mon-red-automat M', tedy automat, který je zároveň stavově minimální i prefixově korektní. Dosažitelnost všech stavů spolu s prefixovou korektností rovněž zaručuje, že každý symbol vstupní abecedy je obsažen v nějakém slově jazyka $L(M')$.

2.3 Popis metody

V této části neformálně představíme metodu, jak pro libovolný normovaný mon-red-automat M rozpoznávající neprázdný jazyk lokalizovat prefixovou a (pokud možno všechny) postprefixové (ne)korektnosti vzhledem k jazyku $L(M)$ v libovolném slově «w» $\in$ «Σ_M^*». Zařízení, které použijeme pro lokalizaci (ne)korektností zde představenou metodou, spolu s implementací metody pomocí tohoto zařízení formálně popíšeme v následující části 2.4.

Během analýzy budeme do pracovního seznamu vkládat nové položky s pomocnými symboly. Použijeme pomocné symboly dvou typů, každý typ slouží jinému účelu, a sice

- vymezení korektních a nekorektních podslov analyzovaného slova,
- přenesení informace o průběhu výpočtu mezi různými etapami.

Pomocným symbolům vymezujícím v analyzovaném slově korektní a nekorektní podslova říkáme *značky*. Značky používáme dvě,

! – pro vymezování nekorektností,

? – pro vymezování nejednoznačností[2].

Lokalizace prefixové (ne)korektnosti. Protože je automat M prefixově korektní, můžeme ho použít pro nalezení nejdelšího korektního prefixu slova «w», a tedy i zleva první nekorektnosti. K tomu dojde v etapě, ve které automat M skončí v koncovém stavu ERR. Výpočet automatu M nad slovem «w» až do této chvíle vypadá takto:

1. Nejprve automat zkrátí slovo «w» na slovo «w'»,
$$«w» \Rightarrow_M^* «w'».$$
2. Pak se v další etapě přesune přes prefix «x slova «w'» do nějakého nekoncového stavu $s \in S_M$,
$$\delta_M^*(s_M, x) = s \in S_M,$$
3. Nakonec ze stavu s přejde přes následující symbol $a \in \Sigma_M \cup \{»\}$ do koncového stavu ERR,
$$\delta_M(s, a) = \texttt{ERR}.$$

[2]Nejednoznačností rozumíme podslovo, o jehož (ne)korektnosti neumíme rozhodnout.

Právě nalezenou prefixovou nekorektnost označíme značkou `!`. Vložíme jí mezi «x a symbol a.

Prefixová korektnost a vlastnost zachování bezchybnosti zaručuje, že automat M v posledním uvedeném kroku navštívil symbol a poprvé. Je-li tedy «w'» = «xay pro nějaké y, pak ay je sufix původního vstupního slova «w». Povšimněme si, že sufix ay (díky monotonii red-automatu M) jednoznačně určuje pozici značky `!` vkládané do slova «w».

Lokalizace postprefixových (ne)korektností. Jak pokračovat s analýzou slova za nalezenou první prefixovou nekorektností, nad sufixem ay slova «w»?

Použijeme funkci δ_M. Tuto funkci jsme zavedli jako rozšíření přechodové funkce f_M. Popisuje přechody nejen mezi jednotlivými stavy, ale i mezi celými podmnožinami množiny $S_M \cup F_M \cup \{\texttt{RED}\}$ všech stavů (koncových i nekoncových) a pomocného stavu `RED`. Díky tomu může současně popsat všechny možné výpočty automatu M nad sufixem ay vstupního slova «w» najednou.

Funkci δ_M nad sufixem $ay = a_0 a_1 \ldots a_{|y|}$ začneme počítat z množiny S_M všech nekoncových stavů automatu M. Dokud je množina

$$S = \delta_M^*(S_M, a_0 \ldots a_i) \setminus \{\texttt{ERR}\}$$

neprázdnou podmnožinou množiny S_M všech nekoncových stavů automatu M, přesuneme se na další položku vpravo a množinu S nahradíme množinou stavů $\delta_M(S, a_{i+1}) \setminus \{\texttt{ERR}\}$.

Důležitý je okamžik (resp. místo ve vstupním seznamu), kdy množina S přestane být neprázdnou podmnožinou S_M. Rozeberme si všechny možné takové situace a jak se v každé z nich zachováme.

Korektní sufix. Množina S obsahuje přijímající koncový stav `ACC`;

$$\texttt{ACC} \in S.$$

To znamená, že sufix analyzovaného slova za poslední vloženou značkou je sufixem nějakého slova jazyka $L(M)$. Žádná další nekorektnost již v analyzovaném slově nemůže být a naše práce nad tímto slovem končí.

Jednoznačná nekorektnost. Množina S je prázdná;

$$S = \emptyset.$$

Všechny možné výpočty automatu M nad sufixem slova za poslední dosud vloženou značkou tedy skončily stavem `ERR`. Objevili jsme sufix korektního jádra analyzovaného slova neboli postprefixovou korektnost. Rovněž jsme identifikovali další nekorektnost tvořenou uvedeným sufixem korektního jádra a symbolem právě snímaným pracovní hlavou. Vlevo před položku pod pracovní hlavou vložíme novou položku se značkou `!`. Další postprefixové (ne)korektnosti hledáme stejným způsobem vpravo za právě vloženou značkou.

Nejednoznačná redukce. S neobsahuje ACC a obsahuje dvě různé redukční koncové stavy automatu M nebo aspoň jeden nekoncový stav a aspoň jeden redukční koncový stav; pro nějakou redukční posloupnost n nastává

$$\mathtt{ACC} \notin S \supsetneq \{\mathtt{RED}(n)\}.$$

Množině S splňující uvedenou podmínku budeme říkat *nejednoznačná* množina.

V této „nejednoznačné“ situaci máme na výběr různá pokračování analýzy – několik různých redukcí analyzovaného slova nebo redukci a současně přesun pracovní hlavy dále doprava. Volbou nevhodného pokračování bychom mohli do analyzovaného slova „zanést“ nové nekorektnosti. Abychom se tomu vyhnuli, označíme místo v pracovním seznamu, kde k nejednoznačnosti došlo, značkou ? a další postprefixové nekorektnosti hledáme stejným způsobem za právě vloženou značkou. (Značku ? vkládáme vpravo za právě snímanou položku, a to ve dvou krocích, abychom stejné místo neoznačovali vícekrát. Detailní popis uvádíme v části 2.4.2.)

Jednoznačná redukce. Množina S obsahuje právě jednu operaci redukce; pro nějakou redukční posloupnost n nastává

$$S = \{\mathtt{RED}(n)\}.$$

Označme jako u podslovo tvořené symboly vstupní abecedy v položkách za poslední značkou ! nebo ? až k položce snímané pracovní hlavou. Podslovo u je díky normovanosti[3] automatu M podslovem nějakého slova jazyka $L(M)$. Navíc je v libovolném slově tohoto jazyka jehož je podslovem redukováno podle stejné redukční posloupnosti n. Podle této redukční posloupnosti budeme podslovo u nyní redukovat i my, ale vždy jen symboly tohoto podslova, i když je třeba n delší než u.

Že n odstraňuje alespoň jeden symbol podslova u, plyne z jednoznačnosti redukce podslova u. Kdyby tomu tak nebylo, pak by pro nějaké z a n' bylo

$$n = n' \cdot \mathtt{0}^{|u|} \quad \text{a} \quad \delta_M^*(s_M, zu) = \mathtt{RED}(n).$$

Nechť z' je nejkratší takové z. Protože je M prefixově korektní, je «$z'u$ prefixem nějakého slova jazyka «$L(M)$». V následující etapě nastává jedna z těchto variant:

$$\begin{aligned}
&\delta_M^*(s_M, (z'/n') \cdot u) = \mathtt{RED}(n'') \quad \text{pro nějaké } n'' \\
&\delta_M^*(s_M, (z'/n') \cdot u) = \mathtt{RED} \\
&\delta_M^*(s_M, (z'/n') \cdot u) = \mathtt{ACC} \\
&\delta_M^*(s_M, (z'/n') \cdot u) \in S_M
\end{aligned}$$

Každá z uvedených variant je ale v rozporu s jednoznačností redukce podslova u. Pokud nastává první z nich, pak n'' redukuje pouze symboly podslova u (protože z' je nejkratší), takže $n'' \neq n$ a $\mathtt{RED}(n'') \in S \not\subseteq \{\mathtt{RED}(n), \mathtt{ERR}\}$. U zbývajících variant je rozpor s jednoznačností redukce podslova u zřejmý.

Kromě redukce u podle n ještě do seznamu vložíme položku s pomocným symbolem – množinou U dvojic tvořených nekoncovým stavem automatu M a slovem nad abecedou Σ_M délky menší než charakteristická konstanta k_M automatu M.

[3] konkrétně díky dosažitelnosti každého stavu a prefixové korektnosti

Tato množina poslouží v následující etapě k úpravě množiny stavů počítané funkcí δ_M. Cílem je zabránit situaci, kdy pro nějaké $s \in S_M$ nastává

$$\delta_M^*(s, u) = \texttt{ERR} \neq \delta_M^*(s, u/n).$$

Takové nekoncové stavy s musíme eliminovat. Množinu U určíme takto:

- Je-li $|u| \geq |n|$, pak provedeme redukci pracovního seznamu podle redukční posloupnosti n a před první redukovanou položku vložíme položku s množinu U obsahující dvojice (s, λ) tvořené všemi nekoncovými stavy automatu M, ve kterých jsme se v právě probíhající etapě přesunuli přes prefix u_1 podslova u délky $|u| - |n|$ nad první redukovanou položku se symbolem vstupní abecedy, a ze kterých se po přesunu přes sufix u_2 podslova u délky $|n|$ automat M dostane do stavu $\texttt{RED}(n)$;

$$U = \{(s, \lambda) \mid \exists s' \in S_M : \delta_M^*(s', u_1) = s \text{ a } \delta_M^*(s, u_2) = \texttt{RED}(n)\}.$$

- Je-li $|u| < |n|$, pak by redukce pracovního seznamu podle n zasáhla i část seznamu před u; tam by se redukovalo podle prefixu n_1 redukční posloupnosti n délky $|n|-|u|$. Část před posledním ! nebo ? ale může být v nějakém slově jazyka $L(M)$ redukována jiným způsobem nebo také vůbec ne. Proto budeme redukovat jen položky za posledním ! nebo ?. Současně s redukcí také bezprostředně za poslední ! nebo ? vložíme novou položku obsahující množinu definovanou takto:

$$U = \{(s, x) \mid \exists v \in \Sigma_M^* : x = v/n_1 \text{ a } |v| = |n_1| \text{ a } \delta_M^*(s, vu) = \texttt{RED}(n)\}.$$

Vložení množiny U do pracovního seznamu je důležité pro zajištění návaznosti po sobě jdoucích etap. V následující etapě jí použijeme pro zúžení množiny stavů S. Jakmile přesuneme pracovní hlavu na položku obsahující U, nahradíme S množinou

$$S' = \{\delta_M^*(s, v) \mid (s, v) \in U\}.$$

Tak zajistíme, že nad část seznamu zredukovanou v předchozí etapě obsahující podslovo u/n vstoupíme v prvním případě jen v těch stavech, které vedly k uvedené redukci, a v druhém případě jen v těch stavech, do kterých se automat dostane po přechodu přes v/n_1 ze stavů, které vedly k redukci vu/n, pro všechna možná v délky $|n| - |u|$.

Právě uvedeným způsobem konstruujeme a vkládáme množinu U pouze v případě, že bezprostředně před vkládanou položkou s množinou je položka obsahující symbol vstupní abecedy nebo značkový pomocný symbol. Pokud ale obsahuje nějaký neznačkový pomocný symbol – množinu U', pak za u vezmeme slovo tvořené symboly vstupní abecedy v seznamu za U' až k položce snímané pracovní hlavou, $|u| \leq |n|$, a místo vložení provedeme nahrazení množiny U' množinou U spočítanou takto:

$$\begin{aligned} U = \{(s, x) \mid \exists y, (s', x') \in U' : x = yx'/n_1 \text{ a} \\ \delta_M^*(s, y) = s' \text{ a } \delta_M^*(s', x'u) = \texttt{RED}(n) \text{ a} \\ |y| = \max\{0, |n_1| - |x'|\}\}, \end{aligned}$$

kde n_1 je prefix n délky $|n| - |u|$. Ve všech případech je délka slova x obsaženého v libovolné dvojici vkládané množiny U shora omezená charakteristickou konstantou k_M automatu M, takže U je vždy konečná.

Poznámky k implementaci metody lokalizace. K implementaci jednoznačné redukce zařízením s konečnou řídící jednotkou ještě poznamenejme, že v řídící jednotce nelze držet potenciálně neomezeně dlouhé u. Pro naše účely ale úplně stačí pamatovat si sufix pracovního seznamu pod a před pracovní hlavou délky nejvýše $2 \cdot k_M$ spolu s obsahy množiny S v posledních $2 \cdot k_M$ krocích.

Musíme také počítat s tím, že jsme (v předchozích etapách) vložily do pracovního seznamu pomocné symboly, a sice značky `!` nebo `?` nebo neznačkové pomocné symboly. Ty musíme zohlednit nejen při vkládání dalších pomocných symbolů, ale i při přesunech pracovní hlavy zařízení směrem doprava nebo při redukcích.

Podrobněji obsah řídící jednotky zařízení použitého pro implementaci metody spolu s omezením jeho velikosti i zohlednění pomocných symbolů vysvětlíme v části 2.4.2.

Projekce značek do vstupního slova. Značky `!` a `?` vložené při analýze do postupně redukovaného pracovního seznamu můžeme promítnout do původního vstupního seznamu bezprostředně před položky, před kterými se poprvé během analýzy v seznamu objevily. Touto projekcí v původním vstupním slově vymezíme prefixovou (ne)korektnost a následné postprefixové (ne)korektnosti, jak říkají následující tvrzení, která dokážeme v části 2.5:

- Je-li «u`!` prefixem vstupního slova s promítnutými značkami a u neobsahuje značku `!`, pak u neobsahuje ani značku `?` a «u je nejdelší korektní prefix analyzovaného vstupního slova vzhledem k jazyku $L(M)$.

- Je-li `!`u`!` nebo `?`u`!` podslovem vstupního slova s promítnutými značkami a u neobsahuje žádnou značku `!` ani `?`, pak u je sufixem nějakého korektního jádra analyzovaného vstupního slova vzhledem k jazyku $L(M)$.

- Je-li `!`u» nebo `?`u» sufixem vstupního slova s promítnutými značkami a u neobsahuje žádnou značku `!` ani `?`, pak u» je sufixem nějakého slova jazyka $L(M)$.

- Je-li `!`u`?` nebo `?`u`?` podslovem vstupního slova s promítnutými značkami a u neobsahuje žádnou značku `!` ani `?`, pak u je podslovem nějakého slova jazyka «$L(M)$».

Právě popsanou metodu robustní redukční analýzy ještě ukážeme na následujícím příkladu.

Příklad 2.3.1. Podívejme se, jak funguje uvedená metoda pro 3 různé mon-red-automaty M_1, M_2 a M_3 rozpoznávající jazyk zjednodušených aritmetických výrazů. Tyto automaty jsou normované, mají tedy všechny vlastnosti požadované popsanou metodou lokalizace prefixové a postprefixových (ne)korektností. Jejich přechodové funkce jsou zadané tabulkami 2.1, 2.2 a 2.3. Liší se tím, jak redukují podslova `a+`(`+a`)*. Automat M_1 odstraňuje ze zpracovávaného slova neuzávorkovaná podslova `a+` zleva a uzávorkovaná podslova `+a` zprava. Automat M_2 odstraňuje podslovo `+a` bez ohledu na uzávorkování všude zprava. Automat M_3 je ostře monotónní a redukuje podslovo `a+` všude zleva. Metodu lokalizace zde demonstrujeme na slově

«`a++(a)+)(a`».

	a	+	(	)	»
$\Rightarrow s_0$	s_1	ERR	s_2	ERR	ERR
s_1	ERR	RED(11)	ERR	ERR	ACC
s_2	s_3	ERR	s_2	ERR	ERR
s_3	ERR	s_4	ERR	RED(101)	ERR
s_4	s_5	ERR	s_2	ERR	ERR
s_5	ERR	s_4	ERR	RED(110)	ERR

Tabulka 2.1: Přechodová tabulka automatu M_1.

	a	+	(	)	»
$\Rightarrow s_0$	s_1	ERR	s_2	ERR	ERR
s_1	ERR	s_3	ERR	ERR	ACC
s_2	s_4	ERR	s_2	ERR	ERR
s_3	s_5	ERR	s_2	ERR	ERR
s_4	ERR	s_6	ERR	RED(101)	ERR
s_5	ERR	s_3	ERR	ERR	RED(110)
s_6	s_7	ERR	s_2	ERR	ERR
s_7	ERR	s_6	ERR	RED(110)	ERR

Tabulka 2.2: Přechodová tabulka automatu M_2.

Analýzy tohoto slova jsou na obrázcích 2.2, 2.3 a 2.4. Každý obrázek obsahuje i výchozí analyzované slovo s promítnutými značkami ! a ?.

2.4 Implementace metody

V této části představíme nový typ zařízení, *robustní analyzátor*, a ukážeme, jak pomocí tohoto zařízení implementovat metodu lokalizace prefixových a postprefixových (ne)korektností popsanou v předchozí části.

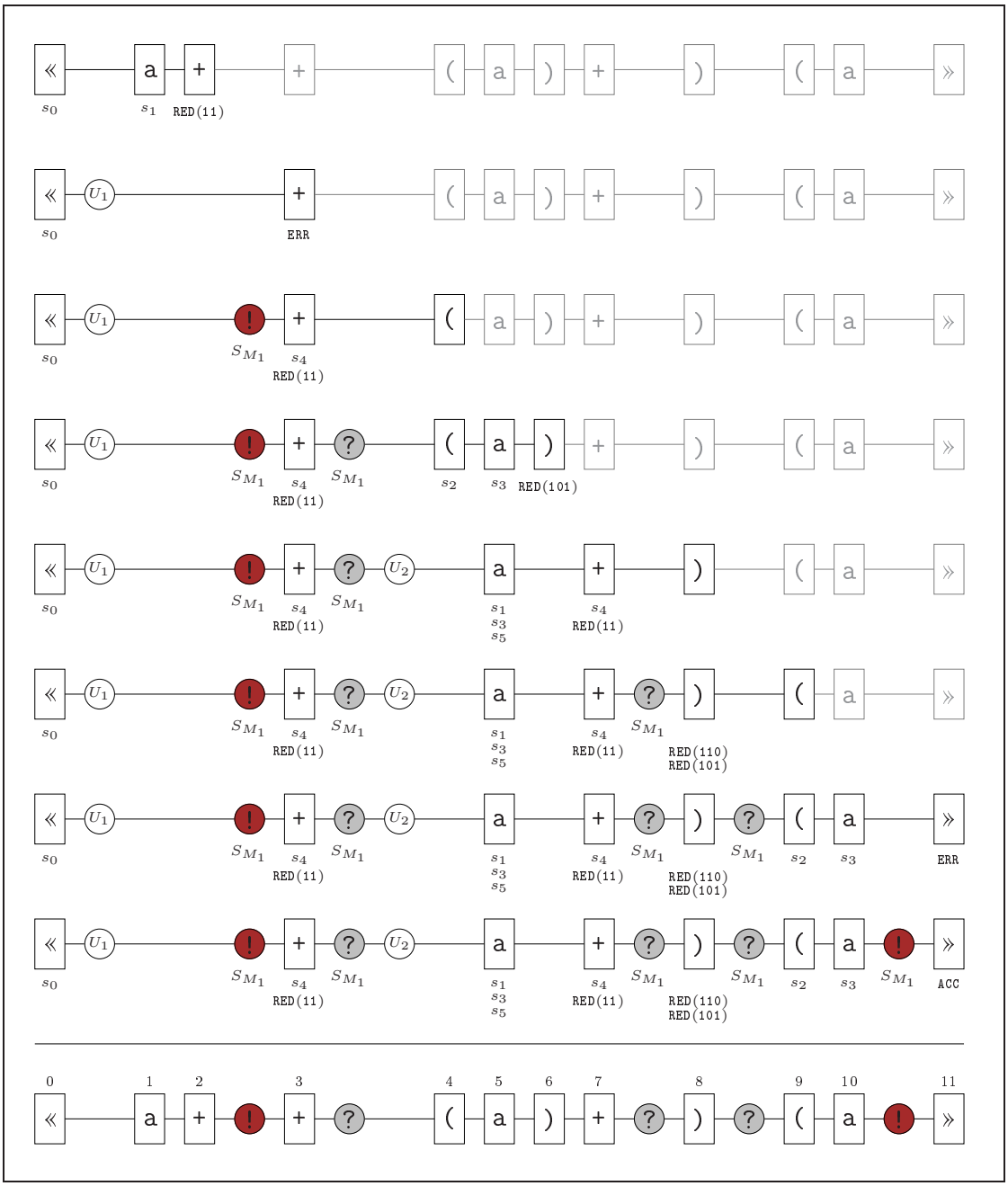
2.4.1 Robustní analyzátor

Robustní analyzátor musí umět redukovat pracovní seznam a vkládat do něho nové položky obsahující značky nebo jiné pomocné symboly.

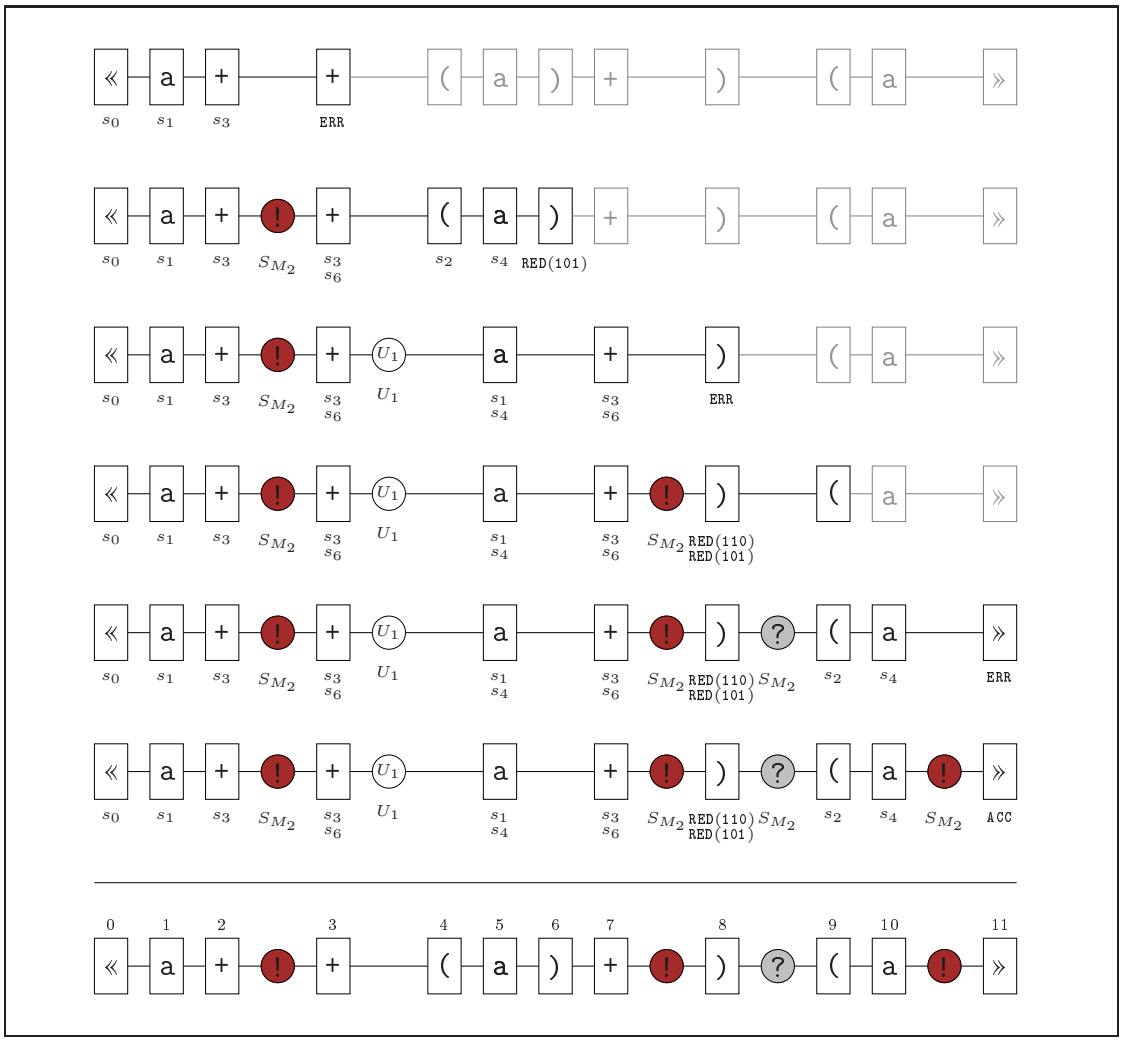
Protože cílem robustního analyzátoru není rozpoznávání nějakého jazyka, ale vyhledávání nekorektností vzhledem k jazyku zadanému normovaným mon-red-automatem, má analyzátor místo koncových stavů ACC a ERR speciální stav END. Ten ukončuje celý výpočet analyzátoru nad daným pracovním seznamem.

	a	+	(	)	»
$\Rightarrow s_0$	s_1	ERR	s_2	ERR	ERR
s_1	ERR	RED(11)	ERR	ERR	ACC
s_2	s_3	ERR	s_2	ERR	ERR
s_3	ERR	RED(11)	ERR	RED(101)	ERR

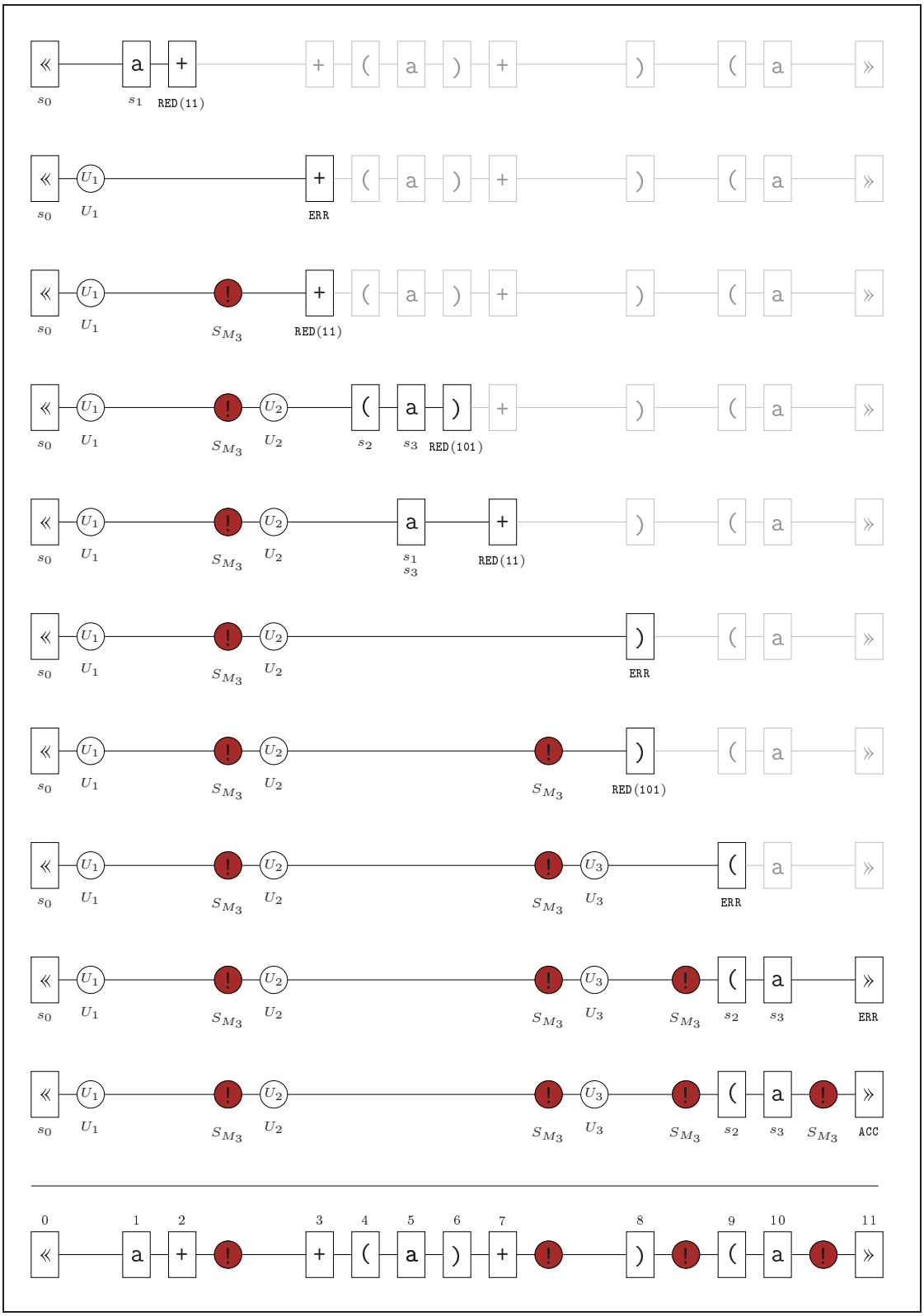
Tabulka 2.3: Přechodová tabulka automatu M_3.



Obrázek 2.2: Analýza slova «a++(a)+)(a» robustním analyzátorem A_1.



Obrázek 2.3: Analýza slova «a++(a)+)(a» robustním analyzátorem A_2.



Obrázek 2.4: Analýza slova «a++(a)+)(a» robustním analyzátorem A_3.

Robustní analyzátor A se, stejně jako redukční automat, skládá z řídící jednotky, která se nachází v jednom z konečně mnoha stavů z množiny S_A, a z pracovní hlavy, která spojuje řídící jednotku se seznamem položek a snímá symbol v jedné z nich. Seznam je ohraničen levým a pravým omezovačem « a ». Všechny ostatní jeho položky obsahují symboly z konečné *vstupní abecedy* X_A nebo z konečné *pomocné abecedy* Y_A. Tyto dvě abecedy jsou navzájem disjunktní a neobsahují omezovač « ani ». Stav řídící jednotky určuje přechodová funkce. Nový stav počítá ze stavu současného a ze symbolu v položce seznamu snímané pracovní hlavou.

Formální odlišnost je i v oddělení redukcí od přechodové funkce prostřednictvím redukční funkce. Vkládání pomocných symbolů do seznamu předepisuje další, vkládací funkce. Chování analyzátoru je tedy určeno místo jedné hned třemi funkcemi:

Přechodová funkce

$$t_A : S_A \times (X_A \cup Y_A) \longrightarrow S_A$$

určuje nový stav, do kterého přejde řídící jednotka ze současného stavu po přečtení symbolu v položce seznamu snímané pracovní hlavou.

Redukční funkce

$$r_A : S_A \longrightarrow R_A, \qquad R_A \subseteq (\mathtt{1} \cdot \mathtt{0}^*)^*$$

přiřazuje novému stavu řídící jednotky *redukční posloupnost* z (konečné) množiny R_A. Redukční posloupnost v případě robustního analyzátoru interpretujeme stejně jako u redukčního automatu. Jedinou vyjímkou je prázdná redukční posloupnost λ, která ponechává pracovní seznam beze změny.

Vkládací funkce

$$i_A : S_A \longrightarrow I_A, \qquad I_A \subseteq (Y_A \cdot \mathtt{0}^*)^*$$

přiřazuje novému stavu řídící jednotky *vkládací posloupnost* z (konečné) množiny I_A. Vkládací posloupnost je obdobou redukční posloupnosti. Říká, které pomocné symboly se mají vložit do seznamu, a kam (relativně vzhledem k poloze pracovní hlavy) se mají vložit. Je-li hodnotou vkládací funkce prázdná posloupnost λ, nevkládá se nikam nic. Je-li její hodnotou `!`, vkládá se nová položka se značkou `!` vlevo před položku snímanou pracovní hlavou. Hodnota `?000` říká, že se má vložit značka `?` před třetí položku vlevo před pracovní hlavou.

Krok výpočtu robustního analyzátoru. Všechny tři funkce se podílejí na každém kroku výpočtu analyzátoru. Ten se skládá z následujících akcí:

Přesun doprava. Analyzátor začíná každý svůj krok přesunem pracovní hlavy doprava na následující položku pracovního seznamu.

Aplikace přechodové funkce. Podle aktuálního stavu řídící jednotky a symbolu snímaného pracovní hlavou analyzátor určí pomocí přechodové funkce nový stav řídící jednotky. Je-li nový stav `END`, pak výpočet končí.

Aplikace redukční funkce. Podle nového stavu řídící jednotky určí analyzátor redukční posloupnost. Pak ze seznamu odstraní položky, jak to redukční posloupnost předepisuje.

Aplikace vkládací funkce. Podobně jako v případě redukční funkce analyzátor podle nového stavu řídící jednotky určí vkládací posloupnost a vloží podle ní do pracovního seznamu nové položky obsahující pomocné symboly.

Stejně jako redukční automat, i robustní analyzátor zpracovává seznam v etapách. Etapa končí, jakmile analyzátor dosáhne stavu `END` nebo dojde ke změně pracovního seznamu podle neprázdné hodnoty redukční nebo vkládací funkce. Poté hned přepne svou řídící jednotku do počátečního stavu s_A a přesune pracovní hlavu na začátek seznamu, nad první položku obsahující levý omezovač «. Stejně jako pro redukční automat můžeme tedy i pro robustní analyzátor definovat monotonii i relaci $\Rightarrow_A$.

2.4.2 Postprefixový robustní analyzátor

Nyní ukážeme, jak popsanou metodu lokalizace prefixových a postprefixových korektností implementovat robustním analyzátorem. Zkonstruujeme ho pro libovolný normovaný mon-red-automat M rozpoznávající neprázdný jazyk. Získaný analyzátor A budeme nazývat *postprefixový robustní analyzátor*.

Analyzátor nejprve nalezne nejdelší korektní (vzhledem k $L(M)$) prefix slova obsaženého v pracovním seznamu, označí ho značkou `!` a dále pak bude hledat postprefixové (ne)korektnosti jazyka $L(M)$ tak, že bude pomocí funkce δ_M (počítané pro celé množiny stavů) simulovat všechny možné výpočty automatu M na částech pracovního seznamu za značkou `!` označující chybu nebo za značkou `?` označující nemožnost jednoznačného pokračování simulace výpočtu automatu M.

Vstupní abeceda X_A analyzátoru se shoduje se vstupní abecedou Σ_M automatu M. Pomocná abeceda Y_A je tvořena značkami `!`, `?` a množinami dvojic stavů automatu M a slov nad abecedou Σ_M délky menší než k_M. Stavy řídící jednotky analyzátoru jsou speciální stavy `red`, `END` nebo trojice

$$\langle m, u, \gamma \rangle$$

tvořené redukční posloupností m, slovem u složeným ze vstupních symbolů ukončeným případně i pravým omezovačem » a řetězcem $\gamma \in Y_A^*$ pomocných symbolů. Délka m je omezená konstantou $2 \cdot k_M$, délka u i γ konstantou k_M. Počátečním stavem s_A analyzátoru je trojice

$$\langle \lambda, \lambda, \{(s_M, \lambda)\} \rangle.$$

Funkce postprefixového robustního analyzátoru. Přechodová, redukční a vkládací funkce postprefixového robustního analyzátoru jsou definované v tabulce 2.4. První sloupec stručně popisuje krok analyzátoru, druhý sloupec obsahuje hodnoty přechodové funkce pro daný krok, třetí sloupec hodnoty vkládací a redukční funkce, čtvrtý sloupec udává podmínku, která musí být pro daný krok splněna. Redukční resp. vkládací funkci uvádíme pouze v případech, kdy vrací neprázdnou redukční resp. vkládací posloupnost.

popis kroku	přechodová funkce	vkládací a redukční funkce	podmínka pro $S' = \delta_M(S, a) \setminus \{\texttt{ERR}\}$
přechod přes vstupní symbol	$t_A(\langle m, u, \gamma{\cdot}U\rangle, a) = \langle m'{\cdot}\texttt{0}, u'{\cdot}a, \gamma'{\cdot}U'\rangle$		$\emptyset \neq S' \subseteq S_M$
přechod přes pomocný symbol	$t_A(\langle m, u, \gamma{\cdot}U\rangle, V) = \langle m'{\cdot}\texttt{1}, u, \gamma{\cdot}V\rangle$		
jednoznačná nekorektnost	$t_A(\langle m, u, \gamma{\cdot}U\rangle, a) = \langle \lambda, \lambda, \emptyset\rangle$	$i_A(\langle \lambda, \lambda, \emptyset\rangle) = \texttt{!}$	$S' = \emptyset$
přechod přes značku !	$t_A(\langle m, u, \gamma{\cdot}U\rangle, \texttt{!}) = \langle \lambda, \lambda, U_M\rangle$		
nejednoznačná redukce	$t_A(\langle m, u, \gamma{\cdot}U\rangle, a) = \langle \lambda, \lambda, U'\rangle$		$S' \supsetneq \{\texttt{RED}(n)\}$
	$t_A(\langle \lambda, \lambda, U'\rangle, b) = \texttt{red}$	$i_A(\texttt{red}) = \texttt{?}$	
(přechod přes značku ?)	$t_A(\langle \lambda, \lambda, U'\rangle, \texttt{?}) = \langle \lambda, \lambda, U_M\rangle$		
jednoznačná redukce	$t_A(\langle m, u, \gamma{\cdot}U\rangle, a) = \langle m'{\cdot}\texttt{0}, u'{\cdot}a, \gamma'{\cdot}U'\rangle$	$r_A(\langle m'{\cdot}\texttt{0}, u'{\cdot}a, \gamma'{\cdot}U'\rangle) = m_r$ $i_A(\langle m'{\cdot}\texttt{0}, u'{\cdot}a, \gamma'{\cdot}U'\rangle) = m_i$	$S' = \{\texttt{RED}(n)\}$

popis kroku	přechodová funkce	vkládací a redukční funkce	podmínka pro $S' = \delta_M(S, \gg) \setminus \{\texttt{ERR}\}$
korektní sufix	$t_A(\langle m, u, \gamma{\cdot}U\rangle, \gg) = \texttt{END}$		$\texttt{ACC} \in S'$
nejednoznačná redukce	$t_A(\langle m, u, \gamma{\cdot}U\rangle, \gg) = \texttt{END}$		$\texttt{ACC} \notin S' \supsetneq \{\texttt{RED}(n)\}$
jednoznačná nekorektnost	$t_A(\langle m, u, \gamma{\cdot}U\rangle, \gg) = \langle \lambda, \lambda, \emptyset\rangle$	$i_A(\langle \lambda, \lambda, \emptyset\rangle) = \texttt{!}$	$S' = \emptyset$
jednoznačná redukce	$t_A(\langle m, u, \gamma{\cdot}U\rangle, \gg) = \langle m'{\cdot}\texttt{0}, u'{\cdot}\gg, \gamma'{\cdot}U'\rangle$	$r_A(\langle m'{\cdot}\texttt{0}, u'{\cdot}\gg, \gamma'{\cdot}U'\rangle) = m_r$ $i_A(\langle m'{\cdot}\texttt{0}, u'{\cdot}\gg, \gamma'{\cdot}U'\rangle) = m_i$	$S' = \{\texttt{RED}(n)\}$

Tabulka 2.4: Funkce postprefixového robustního analyzátoru.

Tabulka má dvě části: První část rozebírá situace, kdy pracovní hlava analyzátoru snímá vstupní symbol $a \in \Sigma_M$, pomocný symbol V nebo značku `!`. V případě nejednoznačné redukce, která se skládá ze dvou kroků, je v první části tabulky zahrnutý i druhý krok, kdy hlava analyzátoru snímá následující položku obsahující symbol $b \in \Sigma_M \cup \{»\}$ nebo značku `?`. Množinu S z podmínky ve čtvrtém sloupci tabulky a množiny S', U' a U_M z definice přechodové funkce spočítáme takto:

$$\begin{aligned} S &= \{\delta_M^*(s,x) \mid (s,x) \in U\} \\ S' &= \delta_M(S,a) \setminus \{\texttt{ERR}\} \\ U' &= \{(s',\lambda) \mid s' \in S'\} \\ U_M &= \{(s,\lambda) \mid s \in S_M\} \end{aligned}$$

Druhá část tabulky se věnuje situacím, kdy pracovní hlava snímá poslední položku seznamu obsahující pravý omezovač ».

V každé situaci uvedené v tabulce je řídící jednotka ve stavu

$$\langle m, u, \gamma \cdot U \rangle,$$

kde pro množinu S odvozenou z množiny U platí $\emptyset \neq S \subseteq S_M$, m je redukční posloupnost, u je slovo tvořené symboly vstupní abecedy a γ je řetězec tvořený symboly pomocné abecedy. Tento stav řídící jednotky zcela popisuje sufix slova v pracovním seznamu vlevo před pracovní hlavou: Je-li ω nejdelší slovo, které

- je sufixem slova obsaženém v části seznamu vlevo před pracovní hlavou,
- obsahuje nejvýše k_M symbolů vstupní abecedy,
- neobsahuje žádnou značku ani omezovač,

pak redukční posloupnost m odstraňuje z ω všechny pomocné symboly a slovo u se skládá ze všech vstupních symbolů ve slově ω, čili $u = \omega/m$. Řetězec $\gamma \cdot U$ obsahuje neprázdné množiny dvojic, jejichž první složky obsahují nekoncové stavy automatu M, ve kterých robustní analyzátor simuloval přesun automatu M nad pořadím odpovídající vstupní symbol slova u.

Nový stav řídící jednotky je v tabulce 2.4 dán redukční posloupností m', slovem u' a řetězcem γ'. Volíme je tak, aby i pro nový stav řídící jednotky analyzátoru platil uvedený vztah se sufixem slova obsaženého v části pracovního seznamu pod a vlevo před pracovní hlavou.

Podrobněji se ještě podívejme na nejednoznačnou a jednoznačnou redukci.

Nejednoznačná redukce. Nejednoznačná redukce se skládá ze dvou kroků. V prvním z nich postprefixový robustní analyzátor převede řídící jednotku do stavu $\langle \lambda, \lambda, U' \rangle$ a přesune svou pracovní hlavu za symbol, který nejednoznačnou redukci „způsobil“. V druhém kroku pak zkontroluje, zda nejednoznačnou redukci na tomto místě v seznamu již neoznačil v některé z předchozích etap svého výpočtu. Pokud k označení nejednoznačnosti ještě nedošlo (symbol b snímaný pracovní hlavou není značka `?`), přejde do speciálního stavu `red` a nejednoznačnost označí značkou `?`. Pokud k označení nejednoznačnosti již došlo (pracovní hlava snímá `?`), pak pokračuje v analýze za značkou `?` simulací všech možných výpočtů automatu M vycházejících z množiny S_M všech jeho nekoncových stavů.

Jednoznačná redukce. Je-li $S' = \{\texttt{RED}(n)\}$ pro nějakou redukční posloupnost n, pak analyzátor redukuje pracovní seznam a vkládá do něho pomocný symbol zajišťující návaznost následující etapy výpočtu nad zkráceným slovem.

Analyzátor redukuje jen symboly obsažené v sufixu ω slova pod a vlevo před pracovní hlavou popsaném novým stavem řídící jednotky

$$\langle m' \cdot \texttt{0}, u' \cdot a, \gamma' \cdot U' \rangle,$$

a to i v případě, že redukční posloupnost n je delší než slovo $u' \cdot a$. Analyzátor totiž redukuje vždy jen symboly za poslední značkou ! nebo ? nebo levým omezovačem «.

Nechť k je minimum z délek slova $u' \cdot a$ a redukční posloupnosti n. Nechť dále

- u_1 je sufix délky k slova $u' \cdot a$,
- U_1 je k-tý symbol řetězce γ' zprava,
- $n = n_2 n_1$, kde $|n_1| = k$, a
- m_1 je nejdelší sufix posloupnosti $\texttt{0}^k \cdot m' \cdot \texttt{0}$ obsahující právě k symbolů $\texttt{0}$[4].

Redukci analyzátoru rozdělíme do dvou kroků: Nejdříve z ω odstraní všechny pomocné symboly podle redukční posloupnosti m_1. Poté slovo ω/m_1 zredukuje podle redukční posloupnosti n_1. Redukční posloupnost m_r z tabulky 2.4 kombinuje redukční posloupnosti m_1 a n_1 do jedné se stejným efektem, jaký má postupná redukce nejdříve podle m_1 a pak podle n_1.

Nakonec analyzátor vloží do pracovního seznamu pomocný symbol U_1' bezprostředně před vstupní symboly, které v seznamu zůstaly z u_1. Vkládací posloupnost m_i z tabulky 2.4 je tedy rovna $U_1' \cdot \texttt{0}^{|u_1/n_1|}$. Množinu U_1' definujeme takto:

$$\begin{aligned} U_1' = \big\{(s', x') \mid \exists v \in \Sigma_M^*, (s, x) \in U_1 : x' = vx/n_2 \text{ a} \\ \delta_M^*(s, v) = s' \text{ a } \delta_M^*(s', xu_1) = \texttt{RED}(n) \text{ a} \\ |v| = \max\{0, |n_2| - |x|\}\big\}. \end{aligned}$$

Je-li $k = |n|$, pak část seznamu zasažená redukční posloupností n neobsahuje žádnou značku. Slovo v v definici množiny U_1' je prázdné, $u_1 = ua$, $n_2 = \lambda$, takže $s = s'$, $x = x'$ a množina U_1' v tomto případě obsahuje právě všechny dvojice (s, x) množiny U_1 takové, že $\delta_M(s, xu'a) = \texttt{RED}(n)$. (Neobsahuje pouze ty dvojice (s, x) z množiny U_1, pro které je $\delta_M^*(s, xu'a) = \texttt{ERR}$.) V následující etapě vyjde analyzátor z množiny U_1' při počítání funkce δ_M^* nad zredukovaným sufixem seznamu obsahujícím $(u'a/n) \cdot \ldots$.

Je-li $k < |n|$, pak by redukce podle celé redukční posloupnosti n zasáhla i část seznamu před poslední značkou vloženou analyzátorem do seznamu. Tato část seznamu ale nemá žádnou souvislost s částí seznamu za značkou. Proto můžeme redukovat jen symboly za poslední značkou. Také chceme správně omezit množinu stavů automatu M, ve kterých v následující etapě vstoupíme nad zredukovanou část seznamu. Vezmeme tedy všechny dvojice (s', x') takové, že automat M přejde přes nějaké slovo v délky $\max\{0, |n_2| - |x|\}$ do stavu nějaké dvojice (s, x) z množiny U_1 a pak dále přes slovo xu_1 do koncového stavu $\texttt{RED}(n)$. Stav $\delta_M^*(s', vx/n_2)$ je pak stav, ve kterém simulovaný automat M v následující etapě vstoupí nad zredukovanou část seznamu za poslední dosud vloženou značkou.

[4] Pokud je vlevo před k-tým symbolem 0 zprava symbol 1, pak m_1 zleva začíná tímto symbolem 1.

Projekce značek do vstupního slova. Vstupní slovo w s promítnutými značkami vloženými do seznamu během analýzy robustním analyzátorem A budeme označovat

$$p_A(w).$$

Příklad 2.4.1. Lokalizaci prefixových a postprefixových (ne)korektností ve slově «`a++(a)+)(a`» a promítnutí značek do tohoto slova jsme již ukázali v příkladu 2.3.1. Zde pro ilustraci funkce p_A ještě uvádíme její hodnoty nad vstupním slovem «`a++(a)+)(a`» pro robustní analyzátory A_1, A_2 a A_3 sestrojené po řadě pro automaty M_1, M_2 a M_3:

$$p_{A_1}(\text{«}\texttt{a++(a)+)(a}\text{»}) = \text{«}\texttt{a+!+?(a)+?)?(a!}\text{»}$$
$$p_{A_2}(\text{«}\texttt{a++(a)+)(a}\text{»}) = \text{«}\texttt{a+!+(a)+!)?(a!}\text{»}$$
$$p_{A_3}(\text{«}\texttt{a++(a)+)(a}\text{»}) = \text{«}\texttt{a+!+(a)+!)!(a!}\text{»}$$

Věta 2.4.2. *Nechť M je libovolný normovaný mon-red-automat rozpoznávající neprázdný jazyk. Postprefixový robustní analyzátor A automatu M je monotónní.*

Důkaz. Analyzátor je deterministický, takže se nad stejným prefixem analyzovaného slova chová vždy stejným způsobem. Rozeberme si tedy jeho chování v místě seznamu s první odlišnou položkou oproti předchozí etapě.

Z tabulky 2.4 vidíme, že část seznamu změněná oproti předchozí etapě vždy začíná vloženou položkou. Tato vložená položka obsahuje značku `!` nebo `?` nebo pomocný symbol U.

Pokud nová položka obsahuje značku, pak analyzátor v předchozí etapě navštívil ještě následující položku obsahující vstupní symbol nebo pravý omezovač », ale žádnou další již ne. Analyzátor A vždy přesune svou hlavu za položku obsahující značku `!` nebo `?`, takže v tomto případě je jeho výpočet monotónní.

Obsahuje-li nová položka pomocný symbol U, pak v předchozí etapě došlo k jednoznačné redukci. Při té analyzátor A odstranil ze seznamu současně

- položku s neznačkovým pomocným symbolem bezprostředně před první odstraněnou položkou obsahující vstupní symbol,
- všechny položky s neznačkovými pomocnými symboly kdekoli za první odstraněnou položkou,
- položky se vstupními symboly podle (sufixu) redukční posloupnosti dané jednoznačnou redukcí automatu M,

a bezprostředně před první redukovanou položku vložil množinu U. Z libovolného stavu $\delta^*_M(s, x)$, kde $(s, x) \in U$, přejde M (protože je monotónní) přes všechny položky navštívené v předchozí etapě, které zůstaly v seznamu. I v tomto případě je tedy výpočet analyzátoru A monotónní. □

2.5 Záruky metody

V této části poskytneme záruky pro lokalizaci prefixových a postprefixových (ne)korektností metodou představenou v této práci. Nejprve ukážeme význam značek `!` a `?`. Dále vymezíme podtřídu normovaných mon-red-automatů, které

jsou redukčně jednoznačné, a zde uvedená metoda se v jejich případě obejde bez značek ?. Nakonec prodiskutujeme časovou složitost robustní analýzy implementované postprefixovým robustním analyzátorem a ukážeme, jak jí zefektivnit implementací zásobníkovým převodníkem.

2.5.1 Význam značek ! a ?

Význam značek ! a ? promítnutých do analyzovaného vstupního slova udává následující věta. Rozebírá, jaká podslova analyzovaného vstupního slova značky vymezují.

Věta 2.5.1. *Nechť M je normovaný mon-red-automat rozpoznávající neprázdný jazyk a A je jeho postprefixový robustní analyzátor. Pro libovolné slovo «w» $\in$ «Σ^*» platí následující tvrzení:*

1. *Analyzátor A přečte (v jedné nebo více etapách) celé slovo «w» a skončí ve speciálním stavu* END.
2. *Pokud p_A(«w») neobsahuje žádnou značku* !, *pak p_A(«w») = «w» a w je slovem jazyka $L(M)$.*
3. *Pokud «w» $\in$ «$L(M)$», pak p_A(«w») = «w».*
4. *Je-li «u*! *prefixem slova p_A(«w») a u neobsahuje značku* !, *pak u neobsahuje ani značku* ? *a «u je nejdelší korektní prefix slova «w» vzhledem k jazyku $L(M)$.*
5. *Je-li* !u! *nebo* ?u! *podslovem slova p_A(«w») a u neobsahuje žádnou značku* ! *ani* ?, *pak u je sufixem nějakého korektního jádra slova «w» vzhledem k jazyku $L(M)$.*
6. *Je-li* !u» *nebo* ?u» *sufixem slova p_A(«w») a u neobsahuje žádnou značku* ! *ani* ?, *pak u» je sufixem nějakého slova jazyka «$L(M)$».*
7. *Je-li* !u? *nebo* ?u? *podslovem slova p_A(«w») a u neobsahuje žádnou značku* ! *ani* ?, *pak u je podslovem nějakého slova jazyka «$L(M)$».*

Důkaz. Podívejme se na stav řídící jednotky analyzátoru A. Tím je na začátku každé etapy počáteční stav

$$s_A = \langle \lambda, \lambda, \{(s_M, \lambda)\} \rangle,$$

kde s_M je počáteční stav automatu M. Ve většině ostatních případů (kromě kroku, který vkládá značku ?) je v řídící jednotce analyzátoru trojice

$$\langle m, u, \gamma \cdot U \rangle,$$

kde $U \subseteq \big((S_M \cup F_M) \setminus \{\mathtt{ERR}\}\big) \times \Sigma_M^{k_M}$. Pro přesun pracovní hlavy analyzátoru přes pracovní seznam zleva doprava je z tohoto stavu důležitá pouze množina U. Ostatní informace držené v řídící jednotce analyzátor využívá pouze při jednoznačné redukci pracovního seznamu. Nechť $S = \{\delta_M^*(s, x) \mid (s, x) \in U\}$. Pokud $\emptyset \neq S \subseteq S_M$, pak analyzátor přesune svou pracovní hlavu na následující položku

vpravo a podle symbolu, který tato položka obsahuje změní stav své řídící jednotky. Je-li tímto symbolem $a \in \Sigma_M \cup \{»\}$, pak novým stavem řídící jednotky je

$$\langle m', u', \gamma' \cdot U' \rangle, \quad \text{kde} \quad U' = \{(s', \lambda) \mid \exists s \in S : \delta_M(s, a) = s' \neq \texttt{ERR}\},$$

pro nějaká m', u' a γ'. Analyzátor A tak prostřednictvím funkce δ_M a množiny na konci třetí složky stavu své řídící jednotky simuluje výpočty automatu M.

Nejprve proberme chování analyzátoru do vložení značky !. Z toho vyplyne platnost tvrzení 2 a 3 a částečně (pro všechna «w» $\in$ «$L(M)$») i platnost tvrzení 1. Neznačkové pomocné symboly vložené během jednoznačných redukcí do seznamu, který neobsahuje žádnou značku !, neovlivňují nijak množinu nekoncových stavů S odvozenou z množiny U obsažené v řídící jednotce analyzátoru, protože je tato množina jednoprvková a shoduje se s množinou ve snímané položce. Ze stejného důvodu nemůže před vložením značky ! dojít k žádné redukční nejednoznačnosti a tedy ani k vložení značky ?. Z definice přechodové, redukční a vkládací funkce postprefixového robustního analyzátoru (viz tabulku 2.4) snadno nahlédneme, že analyzátor až do vložení značky ! redukuje ze seznamu stejné položky se vstupními symboly jako automat M. Jistě také platí, že analyzátor A ukončí výpočet nad daným seznamem neobsahujícím žádnou značku ! přechodem do speciálního stavu END, právě když automat M přejde nad seznamem obsahujícím stejné vstupní symboly do koncového přijímajícího stavu ACC. Tím jsme dokázali tvrzení 2 a 3 a pro všechna «w» $\in$ «$L(M)$» i tvrzení 1.

Pro důkaz tvrzení 4 stačí k dosud provedenému rozboru doplnit vztah mezi etapou analyzátoru A nad seznamem obsahujícím slovo «xby, $b \in \Sigma_M \cup \{»\}$, ve které vloží do seznamu první značku ! mezi «x a b, a etapou automatu M nad seznamem obsahujícím «xby bez pomocných symbolů. Zřejmě platí, že prefix «xb slova v pracovním seznamu dovede analyzátor A v jediné etapě do stavu

$$\langle \lambda, \lambda, \emptyset \rangle,$$

právě když automat M nad prefixem «$x'b$ získaným z «xb odstraněním právě všech pomocných symbolů přejde do stavu ERR. Protože je automat M normovaný (a tedy prefixově korektní), je «x' prefixem nějakého slova z «$L(M)$» a «$x'b$ není prefixem žádného slova z «$L(M)$». Protože je M monotónní a prefixově korektní, nemohl analyzátor A v žádné z případných předchozích etap navštívit položku se symbolem b.

Dále dokážeme tvrzení 5 až 7. Detailně rozebereme jen tvrzení 5. Tvrzení 6 a 7 lze dokázat drobnou modifikací postupu použitého v důkazu tvrzení 5.

Nechť $!u!a$ je podslovem slova $p_A(w)$, $a \in \Sigma_M \cup \{»\}$ a ua neobsahuje žádnou značku. Ukážeme, že zároveň

$$\exists \alpha, \beta : \alpha u \beta \in «L(M)»,$$
$$\forall \alpha, \beta : \alpha u a \beta \notin «L(M)».$$

Předpokládejme, že analyzátor A zpracuje $!ua$ v k etapách, $k \geq 1$. V první z těchto etap přejde přes značku ! ve stavu

$$\langle \lambda, \lambda, U_M \rangle,$$

kde $U_M = \{(s, \lambda) \mid s \in S_M\}$. Za značkou ! tedy simuluje všechny možné přesuny automatu M přes slovo ua. Po přesunu přes nějaký prefix u_1 slova ua se dostane do stavu

$$\left\langle m, v, \gamma \cdot \left\{\mathtt{RED}(n_1)\right\}\right\rangle$$

pro nějaká m, v, γ a redukční posloupnost n_1. V tomto stavu analyzátor redukuje prefix u_1 slova u podle redukční posloupnosti n_1 na nový, kratší[5] prefix v_1. Stejně jako v první etapě nad slovem $!ua$ i v libovolné z následujících etap (kromě poslední z nich) analyzátor redukuje slovo za značkou !. V i-té etapě ($1 \leq i < k$) tedy analyzátor přesune svou pracovní hlavu přes

$$v_i = (v_{i-1} \cdot u_i)/n_i, \tag{2.5}$$

kde

v_{i-1} je slovo navštívené v i-té etapě, které je tvořené vstupními symboly v položkách seznamu navštívených v předchozí, $(i-1)$-ní etapě, $v_0 = \lambda$,

u_i je slovo tvořené vstupními symboly v položkách navštívených poprvé v i-té etapě a

n_i je redukční posloupnost, podle které analyzátor redukuje slovo $v_{i-1}u_i$ v i-té etapě, $n_0 = \lambda$.

Slovo u pak můžeme zapsat jako spojení $v_0 = \lambda$, u_1, u_2, …, u_k,

$$u = v_0 \cdot u_1 \cdot u_2 \cdot \ldots \cdot u_k.$$

Protože je M prefixově korektní, nenavštíví v i-té etapě poslední symbol a. K tomu dojde až v poslední, k-té etapě, kdy analyzátor přejde přes $!v_{k-1}u_k a$ a vloží značku ! před poslední navštívený symbol, tedy a. Po přesunu přes $v_{i-1}u_i$ ($1 \leq i < k$) se analyzátor dostane do „jednoznačné redukce“ dané redukční posloupností n_i. Podle té redukuje $v_{i-1}u_i$. Po redukci vloží do seznamu novou položku s pomocným symbolem – neprázdnou množinou U_i, kterou omezí stavy a zredukovaná podslova omezené délky před $v_i u_{i+1} \ldots u_k$, se kterými simuluje práci automatu M v následující etapě. V tomto důkazu uvažujeme místo U_i jinou množinu V_i přímo za značkou !, která poslouží stejně, ale hned od začátku. Z ní vytěžíme informaci potřebnou pro sestrojení prefixu α, dokládajícího korektnost podslova u vzhledem k jazyku $L(M)$. Množinu V_i definujeme pro libovolné $i \in \{0, \ldots, k-1\}$ takto:

$$\begin{aligned}
V_0 &= \left\{(s, \lambda) \mid s \in S_M\right\} \\
V_i &= \Big\{(s, x) \mid \exists y, (s', x') \in V_{i-1} : x = yx'/n'_i \quad \text{a} \\
&\qquad\qquad \delta^*_M(s, y) = s' \text{ a } \delta^*_M(s', x'v_{i-1}u_i) = \mathtt{RED}(n_i) \quad \text{a} \\
&\qquad\qquad |y| = \max\{0, |n'_i| - |x'|\}\Big\},
\end{aligned}$$

kde n'_i je prefix redukční posloupnosti n_i délky $\max\left\{0, |n_i| - |v_{i-1}u_i|\right\}$.

Tvrzení 2.5.2. *Existují ω, s, x taková, že*

$$\delta^*_M(s_M, \omega) = s \quad a \quad (s, x) \in V_{k-1} \quad a \quad \delta^*_M(s, xv_{k-1}u_k) \in S_M.$$

[5] To, že analyzátor zkrátí slovo za značkou ! resp. ? jsme diskutovali v popisu „jednoznačné redukce“ na straně 23.

Důkaz tvrzení 2.5.2. Fakt, že je množina V_{k-1} neprázdná a že pro nějaký její prvek (s, x) nastává $\delta_M^*(s, xv_{k-1}u_k) \in S_M$ plyne z předpokladu, že analyzátor nad podslovem $!ua$ vykoná k etap a v poslední z nich vloží značku ! mezi u a a. Protože je nekoncový stav s, stejně jako všechny ostatní stavy automatu M, dosažitelný, existuje slovo ω takové, že $\delta_M^*(s_M, \omega) = s$.

Tvrzení 2.5.3. *Pro libovolná $i \in \{1, \ldots, k-1\}$, ω, s, x existují y, s', x' taková, že*

$$\left(\delta_M^*(s_M, \omega) = s \;\; a \;\; (s, x) \in V_i \right) \implies \left(\delta_M^*(s_M, \omega y) = s' \;\; a \;\; (s', x') \in V_{i-1} \right)$$

Důkaz tvrzení 2.5.3. Předpokládejme, že $i \in \{1, \ldots, k-1\}$, ω, s, x jsou libovolná taková, že $\delta_M^*(s_M, \omega) = s$ a $(s, x) \in V_i$. Existence hledaných y, s' a x' je zaručena z definice množiny V_i, která říká, že existují y a $(s', x') \in S_{i-1}$ takové, že $\delta_M^*(s, y) = s'$. Dohromady s předpokladem tedy dostáváme $\delta_M^*(s_M, \omega y) = s'$ a $(s', x') \in V_{i-1}$.

Hledaný prefix α získáme na základě tvrzení 2.5.2 a 2.5.3 takto: Vezmeme libovolné pevné ω z tvrzení 2.5.2 a připojíme k němu postupně řetězce y_{k-2}, $\ldots$, y_1, jejichž existenci nám pro ω a $i \in \{1, \ldots, k-1\}$ zaručuje tvrzení 2.5.3. Dostaneme slovo

$$\alpha = \omega \cdot y_{k-1} \cdot \ldots \cdot y_1.$$

K α přidáme ještě prázdné slovo x_0, které je druhou složkou všech dvojic množiny V_0. Ukážeme, že pro nějaké α je «αu prefixem nějakého slova jazyka «$L(M)$» a že pro libovolné α slovo «αua prefixem žádného slova jazyka «$L(M)$» není. Z definice množin V_i a vztahu (2.5) vyplývá následující redukční analýza automatu M dokládající, že u je podslovem nějakého slova jazyka $L(M)$:

$$\begin{array}{c}
« \overbrace{\omega\, y_{k-1} \ldots y_i \ldots y_2 y_1 x_0}^{\alpha} \cdot \overbrace{v_0 u_1 u_2 u_3 \ldots u_{i+1} \ldots u_k}^{u}\, a » \\
\Downarrow \\
« \omega\, y_{k-1} \ldots y_{i+1} \ldots y_2 x_1 \cdot v_1 u_2 u_3 \ldots u_{i+1} \ldots u_k\, a » \\
\Downarrow \\
\vdots \\
\Downarrow \\
« \omega\, y_{k-1} \ldots y_{i+1} x_i \cdot v_i u_{i+1} \ldots u_k\, a » \\
\Downarrow \\
« \omega\, y_{k-1} \ldots x_{i+1} \cdot v_{i+1} \ldots u_k\, a » \\
\Downarrow \\
\vdots \\
\Downarrow \\
« \omega\, x_{k-1} \cdot v_{k-1} u_k\, a » \\
\\
\delta_M^*(s_M, \omega\, x_{k-1} \cdot v_{k-1} u_k) \in S_M
\end{array}$$

Ještě zbývá dokázat, že ua není podslovem žádného slova jazyka «$L(M)$». Pro důkaz sporem předpokládejme, že pro nějaký prefix α je «αua prefixem nějakého slova z «$L(M)$». Vezměme nejkratší takové α. Protože u je v libovolném

slově jazyka $L(M)$ redukované jednoznačným způsobem v $k-1$ po sobě jdoucích etapách podle redukčních posloupností $n_1, \ldots, n_{k-1}$, je tak redukované i v «$\alpha u a \beta \in$ «$L(M)$» pro nějaké β. Protože je α nejkratší, hned první etapa nad slovem «$\alpha u a \beta$ redukuje část podslova u podle redukční posloupnosti n_1. Dostáváme tedy

$$« \alpha\, u a\, \beta \Rightarrow_M^* « \alpha'\, v_{k-1} u_k a\, \beta.$$

Pak ale pro nějaká α'' a x je $\alpha' = \alpha'' x$ a

$$\big(\delta_M^*(s_M, \alpha''), x\big) \in V_{k-1},$$

takže $\delta_M^*(s_M, \alpha' v_{k-1} u_k a) = \mathtt{ERR}$, což je spor.

Stejným postupem dokážeme platnost tvrzení 5 pro $?u!a$, protože po přechodu přes značku ? se analyzátor dostane do stejného stavu jako po přechodu přes značku !.

První část důkazu tvrzení 5, v níž dokazujeme, že u je podslovem nějakého slova jazyka $L(M)$, použijeme pro důkaz tvrzení 7.

Rovněž důkaz tvrzení 6 je drobnou modifikací první části našeho postupu. Odlišnost spočívá v tom, že slovo u končí pravým omezovačem » a analyzátor se přes něj v k-té etapě přesune do speciálního stavu END ukončujícího výpočet analyzátoru.

Tím máme dokázané i tvrzení 1 pro «w» $\notin$ «$L(M)$». □

2.5.2 Redukční jednoznačnost

Jaké musí být vlastnosti mon-red-automatu M, aby jeho postprefixový robustní analyzátor značku ? nevložil do žádného slova? Tato značka zprava omezuje ta podslova analyzovaného slova, která v různých slovech jazyka $L(M)$ dovedou automat M k redukcím podle různých redukčních posloupností nebo v jednom slově k redukci a v jiném slově k přesunu doprava. Zavedeme pojmy *jednoznačně redukovatelné podslovo* a *redukčně jednoznačný automat* a ukážeme, že postprefixový robustní analyzátor redukčně jednoznačného automatu značku ? nikdy nepoužije.

Podslovo w je redukčním automatem M *redukovatelné*, jestliže pro nějakou redukční posloupnost n je

$$\mathtt{RED}(n) \in \delta_M^*(S_M, w).$$

Podslovo w je redukčním automatem M *jednoznačně redukovatelné*, jestliže pro nějakou redukční posloupnost n je

$$\mathtt{RED}(n) \in \delta_M^*(S_M, w) \subseteq \{\mathtt{RED}(n), \mathtt{ERR}\}.$$

Podslovo, které je redukovatelné, ale není jednoznačně redukovatelné budeme označovat jako *nejednoznačně redukovatelné* podslovo.

Redukční automat nazveme *redukčně jednoznačným*, je-li každé jeho redukovatelné podslovo jednoznačně redukovatelné.

Příklad 2.5.4. Naše automaty M_1 a M_2 z příkladu 2.3.1 rozpoznávající jazyk zjednodušených aritmetických výrazů nejsou redukčně jednoznačné.

podslovo w	)	a)	+	a+	+a+	...
$\delta^*_{M_1}(S_{M_1}, w)$	RED(110) RED(101)	RED(110) RED(101)	RED(11) s_4	RED(11) s_4	RED s_4	...

(a) nejednoznačně redukovatelná podslova

podslovo w	+a)	(a)	«a+
$\delta^*_{M_1}(S_{M_1}, w)$	RED(110)	RED(101)	RED(11)

(b) minimální jednoznačně redukovatelná podslova

Tabulka 2.5: Redukovatelná podslova automatu M_1.

podslovo w	)	a)	»	a»
$\delta^*_{M_2}(S_{M_2}, w)$	RED(110) RED(101)	RED(110) RED(101)	RED(110) ACC	RED(110) ACC

(a) nejednoznačně redukovatelná podslova

podslovo w	+a)	(a)	+a»
$\delta^*_{M_2}(S_{M_2}, w)$	RED(110)	RED(101)	RED(110)

(b) minimální jednoznačně redukovatelná podslova

Tabulka 2.6: Redukovatelná podslova automatu M_2.

Nejprve se podívejme na automat M_1. Všechna jeho nejednoznačně redukovatelná podslova jsou uvedena v tabulce 2.5a. Jejich délku nelze omezit žádnou konstantou, protože pro libovolné $i \geq 0$ je

$$\delta^*_{M_1}(S_{M_1}, (\texttt{+a})^i\texttt{+}) = \{\texttt{ERR}, \texttt{RED(110)}, s_4\}$$

Minimální jednoznačně redukovatelná podslova tohoto automatu jsou pak v tabulce 2.5b.

Redukční automat M_2 má nejednoznačně redukovatelných podslov konečně mnoho. Všechna jsou v tabulce 2.6a. Jeho jednoznačně redukovatelná podslova minimální délky jsou v tabulce 2.6b.

Ostře monotónní red-automat M_3, jehož robustní analyzátor v příkladě 2.3.1 určil všechny chyby, je redukčně jednoznačný; všechna jeho redukovatelná podslova jsou redukovatelná jednoznačně. To je vidět přímo z tabulky 2.3 jeho přechodové funkce f_{M_3}. Sloupce pro přechod přes symboly + a) jsou jediné, které obsahují redukční koncové stavy, a ve sloupci pro přechod přes symbol + resp.) je jediný stav různý od ERR, a sice RED(11) resp. RED(101).

Význam redukční jednoznačnosti pro lokalizaci syntaktických chyb dokládá následující věta.

Věta 2.5.5. *Je-li M redukčně jednoznačný normovaný mon-red-automat rozpoznávající neprázdný jazyk, pak jeho postprefixový robustní analyzátor nevloží značku ? do žádného analyzovaného slova. V libovolném slově z «Σ^*_M» tedy určí prefixovou i všechny postprefixové nekorektnosti vůči jazyku $L(M)$.*

Důkaz. Redukční jednoznačnost automatu M zaručuje, že pro žádný symbol $a \in \Sigma_M \cup \{»\}$ není množina $\delta_M(S_M, a)$ nejednoznačná. □

Poznámka. Vkládání značek `?` můžeme ještě dále eliminovat následující drobnou modifikací konstrukce postprefixového robustního analyzátoru: Z množiny U držené na konci třetí složky stavu jeho řídící jednotky

$$\langle m, u, \gamma \cdot U \rangle$$

můžeme odstranit dvojici $(\mathtt{RED}(n), \lambda)$, pokud redukční posloupnost n končí $\mathtt{0}^{|u|}$. To znamená, že u je tvořené právě všemi vstupními symboly za poslední dosud vloženou značkou `!` resp. `?`, pod nebo vlevo před pracovní hlavou. Redukce podle n nemění slovo u a tudíž nezpůsobí žádnou nejednoznačnost. Protože je automat M monotónní a akceptuje až po přečtení celého slova, dříve nebo později se buď přesune přes u dále doprava nebo provede redukci, která už u změní. A jen tyto možnosti (obsažené již teď v množině S) nás zajímají z hlediska případné nejednoznačnosti pokračování analýzy slova za poslední dosud vloženou značkou.

Kromě odstranění uvedené dvojice z množiny U je také třeba upravit pomocný symbol v seznamu bezprostředně za nejbližší značkou `!` nebo `?` před pracovní hlavou.

Touto drobnou změnou v konstrukci analyzátoru bychom dosáhli následujícího zpřesnění analýzy slova «`a++(a)+)(a`» z příkladu 2.3.1 v případě postprefixového robustního analyzátoru sestrojeného pro automat M_1 resp. M_2:

«`a+!+?(a)+?)` `!` `(a!`» resp. «`a+!+(a)+!)` `!` `(a!`».

Změna oproti analýze bez uvedené modifikace je zvýrazněna zeleně.

2.5.3 Časová složitost

Časovou složitost metody postprefixové robustní analýzy uvedeme pro dvě její implementace: pro implementaci robustním analyzátorem detailně popsanou v části 2.4 a pro implementaci zásobníkovým převodníkem, kterou stručně nastíníme níže.

Časová složitost analýzy robustním analyzátorem

Postprefixový robustní analyzátor A normovaného mon-red-automatu M zanalyzuje libovolné slovo «w» $\in$ «Σ_M^*» délky n v nejvýše n etapách. V každé etapě (kromě té poslední) přitom odstraní alespoň jednu položku. To znamená, že se v první etapě přesune přes nejvýše n položek, ve druhé etapě přes nejvýše $n-1$ položek, atd. Každá redukce mění sufix seznamu pod a vlevo před pracovní hlavou délky nejvýše $2 \cdot k_M$. Ještě předpokládáme, že návrat pracovní hlavy na začátek seznamu, kterým končí jedna a začíná následující etapa, lze provést v jednotkovém čase. Dohromady tedy dostáváme následující horní mez pro počet přesunů pracovní hlavy:

$$2 \cdot k_M \cdot \sum_{i=1}^{n} i = k_M \cdot n \cdot (n+1)$$

Časová složitost analýzy slova délky n postprefixovým analyzátorem je tedy

$$O(n^2).$$

Časová složitost analýzy zásobníkovým převodníkem

Protože je postprefixový robustní analyzátor A monotónní, jak tvrdí věta 2.4.2, můžeme uvedenou časovou složitost snížit použitím zásobníku. Zběžně zde předvedeme, jak metodu implementovat zásobníkovým převodníkem. Ten bude svým výstupem realizovat projekci značek `!` a `?` do analyzovaného vstupního slova. Jeho výstupem tedy bude vstupní slovo s promítnutými značkami. V zásobníku konstruovaného převodníku budeme držet část seznamu vlevo před pracovní hlavou spolu s množinami U z konce třetí složky stavu analyzátoru. Je-li v seznamu vlevo před pracovní hlavou slovo

$$« \, a_1 \, a_2 \, \ldots \, a_l,$$

je v zásobníku

$$« \, U_0 \, a_1 \, U_1 \, a_2 \, U_2 \, \ldots \, U_{l-1} \, a_l \, U_l.$$

V zásobníku držíme i značky `!` a `?`. Zásobník eliminuje nutnost opakovaně přesouvat pracovní hlavu přes všechny položky navštívené v předchozí etapě. Místo odstranění položek seznamu podle redukční posloupnosti, vložení nových položek podle vkládací posloupnosti a návratu pracovní hlavy na začátek seznamu provedeme změnu nejvýše $2 \cdot k_M$ položek na vrcholu zásobníku. To uděláme tak, že nejprve přesuneme měněné položky z vrcholu zásobníku do řídící jednotky, v řídící jednotce na ně aplikujeme redukční a vkládací funkci a to, co v řídící jednotce zbyde, přesuneme zpět do zásobníku. Monotonie analyzátoru A nám zaručuje, že lze do zásobníku přesunout vše, co v řídící jednotce zůstane po aplikaci redukční a vkládací funkce. Značky `!` a `?` přitom vkládáme nejen do zásobníku, ale i do výstupního seznamu převodníku.

Pro popsaný zásobníkový převodník spočítáme časovou složitost takto: Pracovní hlava se přesune přes všech n symbolů vstupního slova «w». Simulace každé redukce nyní zabere nejvýše $2 \cdot k_M$ kroků pro přesun až $2 \cdot k_M$ symbolů z vrcholu zásobníku do řídící jednotky a dalších nejvýše $2 \cdot k_M$ kroků pro přesun symbolů po aplikaci redukční a vkládací funkce z řídící jednotky zpět do zásobníku. Redukovat můžeme nejvýše n symbolů vstupní abecedy v nejvýše n „etapách“. Počet kroků analýzy při použití zásobníku můžeme nyní shora omezit takto:

$$n + n \cdot (2 \cdot k_M + 2 \cdot k_M) = (4 \cdot k_M + 1) \cdot n$$

Časová složitost analýzy slova délky n při použití zásobníku je tedy

$$O(n).$$

Předchozí rozbor shrnujeme v následujícím důsledku.

Důsledek 2.5.6. *Nechť M je libovolný normovaný mon-red-automat a A jeho postprefixový robustní analyzátor. Pak lze sestrojit deterministický zásobníkový převodník, který překládá libovolné slovo «w» $\in$ «Σ_M^*» na slovo p_A(«w»).*

2.6 Shrnutí

Navrhli jsme metodu, kterou lze pro libovolný neprázdný jazyk $L \in \mathsf{DCFL}$ nad abecedou Σ (jejíž každý symbol je obsažen v nějakém slově jazyka L) v libovolném slově $w \in \Sigma^*$ lokalizovat prefixovou (ne)korektnost a *některé* postprefixové (ne)korektnosti, to vše *v lineárním čase* při použití zásobníkového převodníku. Dále jsme určili, za jakých dodatečných podmínek navrženou metodou nalezneme *všechny* postprefixové (ne)korektnosti.

Tam, kde kvůli nejednoznačným redukcím nelze lokalizovat celý sufix korektního jádra v analyzovaném slově w, vymezí navržená metoda ve w značkami ! a ? alespoň podslova, která sice vedou k nejednoznačné redukci, ale jsou zaručeně korektní vzhledem k danému jazyku L.

V této kapitole jsme použili termín zásobníkový převodník. Podrobnosti o zásobníkových převodnících lze nalézt např. v [14].

Kapitola 3

Normalizace redukčních automatů

Metoda postprefixové robustní analýzy představená v kapitole 2 je určená pro normované monotónní redukční automaty. Jedná se o podtřídu redukčních automatů vymezenou takto: Normovaný je libovolný mon-red-automat, který je současně

- prefixově korektní a
- stavově minimální.

V této kapitole ukážeme, jak k libovolnému mon-red-automatu, který přijímá neprázdný jazyk, zkonstruovat ekvivalentní mon-red-automat, který má všechny uvedené vlastnosti a je tedy normovaný.

V první podkapitole nejprve k libovolnému mon-red-automatu sestrojíme ekvivalentní LR(0)-gramatiku. Tuto gramatiku využijeme v druhé podkapitole ke konstrukci ekvivalentního prefixově korektního mon-red-automatu. Poslední, třetí podkapitola ukazuje, jak minimalizovat množinu stavů libovolného red-automatu. Pro nás bude důležité zejména to, že při minimalizaci stavů dojde i k odstranění nedosažitelných stavů. Jejich absence je při konstrukci postprefixového robustního analyzátoru postačující.

3.1 Gramatika mon-red-automatu

V této podkapitole popíšeme, jak pro libovolný mon-red-automat M zkonstruovat bezkontextovou gramatiku G_M, jejíž derivační stromy jednoznačně korespondují s výpočty daného automatu. Ukážeme, že tato gramatika generuje právě jazyk rozpoznávaný automatem M, a také, že je to LR(0)-gramatika.

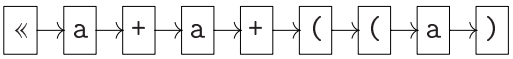
Pro snažší porozumění nejprve zaváděné pojmy předvedeme na monotónním redukčním automatu M, který jsme definovali přechodovou tabulkou 1.1 na straně 14. Tento automat rozpoznává jazyk zjednodušených aritmetických výrazů. Nejdříve však neformálně zavedeme pojmy, o něž se idea konstrukce gramatiky G_M opírá. Jedná se o stromy výpočtů mon-red-automatu.

3.1.1 Stromy výpočtů mon-red-automatu

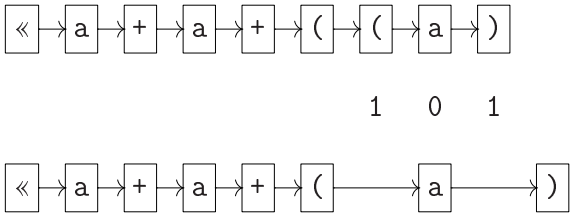
Zde si ukážeme, jak výpočty libovolného mon-red-automatu M popisovat pomocí binárních uspořádaných stromů. Tyto stromy nás postupně dovedou ke gramatice generující právě slova přijímaná automatem M.

Etapa jako seznam navštívených položek. Libovolnou etapu výpočtu redukčního automatu nad daným slovem můžeme jednoznačně určit posloupností symbolů v položkách pracovního seznamu navštívených pracovní hlavou automatu během této etapy. Abychom získali úplnou informaci o etape, stačí k jednotlivým položkám doplnit stavy, ve kterých byly tyto položky navštíveny.

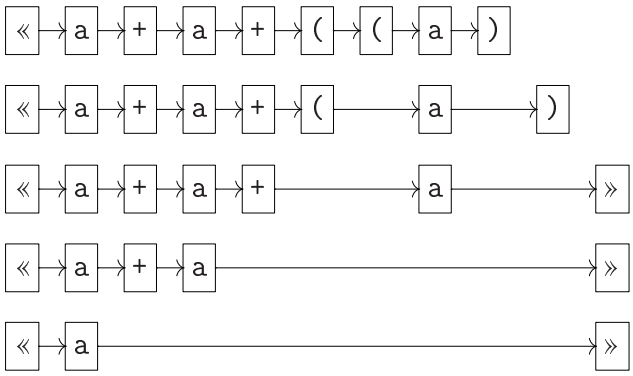
První etapu výpočtu automatu M nad pracovním seznamem, který obsahuje slovo «`a+a+((a))`», zapíšeme tímto způsobem takto:



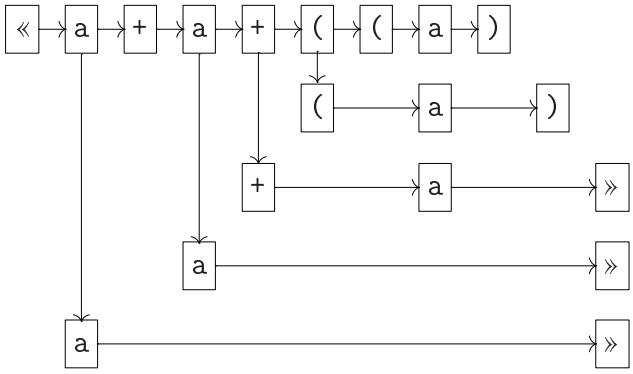
Výpočet jako seznam etap. Výpočet redukčního automatu nad daným slovem pak můžeme zapsat jako seznam takových posloupností určujících po sobě jdoucí etapy tohoto výpočtu. Druhou etapu přidáme pod první etapu tak, aby pod sebou na stejné vertikále ležely symboly ze stejných položek pracovního seznamu a jejich vodorovné pozice tak byly v souladu s redukční posloupností, podle které jsme redukovali pracovní seznam v první etapě:



Celý výpočet automatu M nad slovem «`a+a+((a))`» pak reprezentuje následující uspořádaný seznam po sobě jdoucích etap:



Strom výpočtu. Vidíme, že jednotlivé po sobě jdoucí etapy se shodují až k zleva prvnímu redukovanému symbolu. Shodující se začátky etap sloučíme a z výpočtu v podobě uspořádaného seznamu etap získáme výpočet v podobě uspořádaného stromu.



U takto získaného stromu držíme v platnosti pravidlo použité při kreslení seznamu etap, a sice že všechny vrcholy stromu, kterým odpovídá stejná položka pracovního seznamu, umisťujeme pod sebe na stejnou vertikálu. Maximální cesty, tedy cesty z kořene do listu, odpovídají částem pracovního seznamu navštíveným během jednotlivých etap výpočtu automatu (vrcholy na stejné vertikále uvažujeme pouze jednou). Pořadí synů libovolného vrcholu stromu volíme tak, aby jím určené pořadí maximálních cest odpovídalo pořadí etap výpočtu. Poslední maximální cesta stromu odpovídá přijímající nebo zamítající etapě, všechny ostatní cesty odpovídají etapě, která končí redukcí pracovního seznamu. Strom s právě uvedenými vlastnostmi budeme nazývat *stromem výpočtu* redukčního automatu. Protože redukční automat přijímá až po přečtení celého seznamu, po přesunu přes pravý omezovač », pokrývá strom přijímajícího výpočtu celý pracovní seznam.

Jak je patrné z právě uvedeného obrázku stromu výpočtu, neslučujeme úplně celé shodující se začátky po sobě jdoucích etap. Neslučujeme totiž položku bezprostředně předcházející první položce odstraněné na konci první ze slučovaných etap. Díky tomu je strom výpočtu binární a to nám, jak uvidíme dále, zjednoduší vyjadřování o nahrazování částí výpočtu jinými částmi jiných výpočtů.

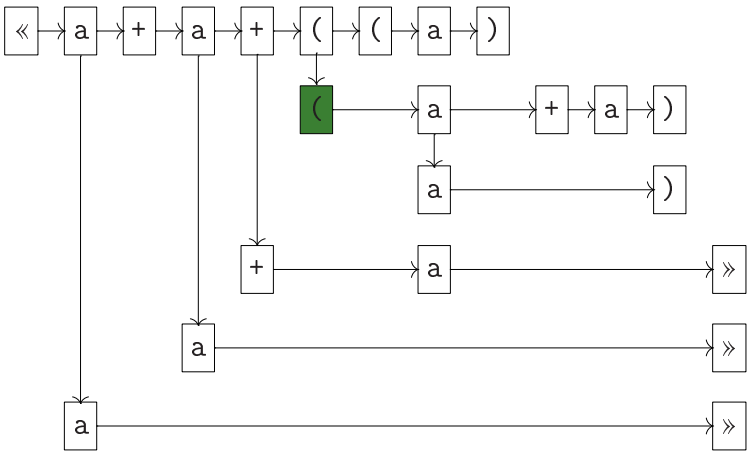
Zaměnitelnost podstromů stromů výpočtů. Až do této chvíle jsme nespoléhali na to, že redukční automat je monotónní. Dále už se ale bez monotonie neobejdeme. Záměnou podstromu stromu výpočtu za jiný podstrom jiného stromu výpočtu získáme strom, který za jistých podmínek reprezentuje nějaký jiný výpočet a je tedy stromem výpočtu daného automatu. To se povede, reprezentuje-li novém stromu každá maximální cesta, tedy cesta z kořene do listu, nějakou etapu výpočtu automatu M a zůstane-li v novém stromu zachována následnost etap představovaných jeho maximálními cestami. Každá maximální cesta stromu výpočtu monotónního redukčního automatu musí ve svých vrcholech obsahovat (v příslušném pořadí) přinejmenším všechny symboly obsažené ve vrcholech předchozí maximální cesty, které neodstraní redukce daná předchozí cestou.

Záměnou podstromu S_1 stromu výpočtu T_1 mon-red-automatu M za podstrom S_2 stromu výpočtu T_2 téhož automatu získáme strom výpočtu tohoto automatu, pokud zároveň

- kořeny podstromů S_1, S_2 obsahují stejný symbol,
- automat M se (libovolné) v etapě procházející ve stromu T_1 kořenem podstromu S_1 dostane po přečtení symbolu v kořeni podstromu S_1 do stejného stavu jako po přečtení symbolu v kořeni podstromu S_2 v libovolné etapě procházející ve stromu T_2 kořenem podstromu S_2,
- symboly na první cestě podstromu S_1 navštívené už i v poslední etapě stromu T_1, která předchází všechny etapy procházející podstromem S_1, se shodují se symboly na první cestě podstromu S_2 navštívenými už i v poslední etapě stromu T_2, která předchází etapám procházejícím podstromem S_2,
- poslední etapa procházející podstromem S_1 resp. S_2 končí v obou případech buď přijetím, nebo zamítnutím, nebo redukcí, která vpravo za vertikálou

kořene podstromu S_1 resp. S_2 zanechá stejné symboly a stejným způsobem (stejnou redukční posloupností) redukuje seznam na vertikále kořene podstromu S_1 resp. S_2 a vlevo před ní.

Zaměnitelnost podstromu ve stromu výpočtu si předvedeme na příkladu následujícího stromu T výpočtu mon-red-automatu M nad slovem «`a+a+((a)+a)`».



Za jaké podstromy je zaměnitelný podstrom S s kořenem ve vrcholu obarveném zeleně?

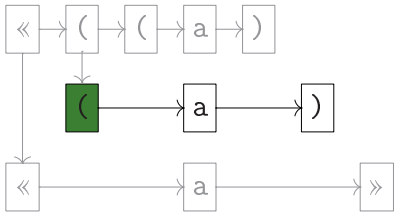
Každá cesta podstromu S z jeho kořene do libovolného z jeho listů navazuje na cestu z kořene celého stromu T do kořene podstromu S a dohromady reprezentuje nějakou etapu výpočtu automatu M. To je dáno tím, že se automat M po přesunu přes symboly vrcholů cesty z kořene stromu T do kořene podstromu S dostane do stavu $\delta_M(s_0, \texttt{a+a+(}) = s_4$ a ze stejného stavu se přesune přes symboly vrcholů maximálních cest podstromu S za jeho kořenem. To samé (se stejným stavem s_4) musí platit pro nový podstrom, kterým ve stromu T nahradíme podstrom S.

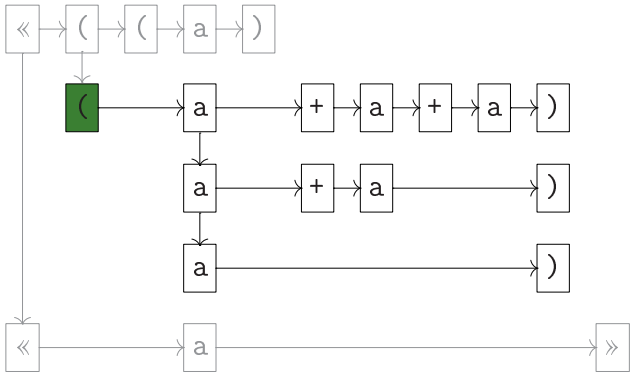
V etapě bezprostředně nad podstromem S navštívil automat M položky pracovního seznamu ležící na vertikále zeleného vrcholu a vpravo za ní, které obsahují podslovo `((a)`. Po redukci, kterou tato etapa končila, zůstalo v seznamu z podslova `((a)` podslovo `(a`. Tímto zredukovaným podslovem musí začínat první kořenová cesta podstromu S i první kořenová cesta libovolného podstromu zaměnitelného ve stromu T za podstrom S. Díky monotonii automatu M obsahuje toto podslovo všechny symboly, které automat navštívil v etapách obsažených ve stromu T nad podstromem S, které zároveň zůstaly po těchto etapách nezredukované v pracovním seznamu. Délka zbylého podslova `(a` je shora omezená konstantou k_M.

Dále musí být i po záměně podstromu zaručena návaznost etapy bezprostředně pod podstromem S. Slovo «`a+a+(a)` v položkách seznamu navštívených v poslední etapě procházející podstromem S rozdělíme vertikálou jeho kořene na dvě části: prefix «`a+a+(` končící symbolem `(` položky na vertikále kořene podstromu a sufix `a)` vpravo za touto vertikálou. Stejně tak rozdělíme i redukční posloupnost `101` obsaženou v redukčním stavu `RED(101)`, kterým končí poslední etapa procházející podstromem S, na `1` a `01`. Pokud bude náhradník podstromu S ve stromu T redukovat prefix «`a+a+(` stejně jako podstrom S, tedy redukční posloupností `1`, a ze slova jeho poslední cesty z kořene do listu zbyde stejné slovo jako v případě podstromu S, tedy `a)/01` = `a`, pak zůstane návaznost etapy bezprostředně pod podstromem S zachována. I zde je důležitá monotonie automatu M a omezení

prefixu redukční posloupnosti a slova, které zbyde z poslední cesty podstromu z jeho kořene do listu, konstantou k_M.

Uvedená kritéria zaměnitelnosti s podstromem S ve stromu T splňují například podstromy následujících stromů výpočtů automatu M s kořeny v zelených vrcholech.





Záměnami tak získáme stromy výpočtů uvažovaného automatu nad slovy «`a+a+((a))`» a «`a+a+((a)+a+a)`».

Zaměnitelnost podstromů stromů výpočtů přirozeně vede k bezkontextové gramatice, jejíž neterminály odpovídají vrcholům stromů výpočtů obohaceným o informace nutné pro reprezentaci etap výpočtu a jejich návaznost (resp. zaměnitelnost podstromů). V libovolném derivačním stromu bezkontextové gramatiky pak můžeme celý podstrom nahradit jiným podstromem jiného derivačního stromu se stejným neterminálem v kořeni. Zbývá ještě dořešit, podle jakých pravidel neterminály přepisovat na terminály a jiné neterminály. Tato pravidla a počáteční neterminál nadefinujeme tak, aby derivační stromy gramatiky jednoznačně odpovídaly stromům přijímajících výpočtů daného mon-red-automatu a aby gramatika generovala stejný jazyk.

3.1.2 Gramatika mon-red-automatu

Pro monotónní redukční automat M sestrojíme gramatiku G_M, jejíž derivační stromy budou úzce korespondovat se stromy výpočtů automatu M.

Grafická reprezentace derivačního stromu. Derivační stromy gramatiky G_M budeme kreslit jinak, než je u derivačních stromů zvykem. Důvodem je to, že chceme, aby tento obrázek derivačního stromu nesl celou informaci o odpovídajícím stromu výpočtu a tedy se i podobně kreslil. Jde hlavně o reprezentaci jednotlivých etap i jejich „dělících míst“. Zachycení dělících míst jednotlivých etap bude také hrát podstatnou úlohu v návrhu pravidel gramatiky G_M. Dělící místa budou zachycovat pravidla se dvěma neterminály na pravé straně.

pravidlo	grafická reprezentace
$X \to aY$	$\begin{array}{c} a \\ \uparrow \\ X \to Y \end{array}$
$X \to Y$	$X \to Y$
$X \to aYZ$	$\begin{array}{c} a \\ \uparrow \\ X \to Y \\ \downarrow \\ Z \end{array}$
$X \to YZ$	$\begin{array}{c} X \to Y \\ \downarrow \\ Z \end{array}$
$X \to a$	$\begin{array}{c} a \\ \uparrow \\ X \end{array}$
$X \to \lambda$	X

Tabulka 3.1: Grafická reprezentace pravidel gramatiky.

V tradičním obrázku derivačního stromu je počáteční neterminál S_0 nahoře a každé ve stromu použité pravidlo je zakresleno tak, že neterminál z levé strany pravidla je nad všemi symboly z pravé strany pravidla, a ty leží na stejné horizontále v pořadí zleva doprava. Příkladem může být následující grafická reprezentace pravidla $X \to aYZ$:

$$\begin{array}{ccccc} & & X & & \\ & \swarrow & \downarrow & \searrow & \\ a & & Y & & Z \end{array}$$

My stejné pravidlo nakreslíme tak, že generovaný terminální symbol a, neterminál X a neterminál Z budou ležet na stejné vertikále pod sebou a neterminál Y bude napravo od X na stejné horizontále. Takto zachytíme dělící místo mezi dvěma etapami příslušným derivačním stromem.

$$\begin{array}{c} a \\ \uparrow \\ X \to Y \\ \downarrow \\ Z \end{array}$$

Celý derivační strom gramatiky G_M pak graficky znázorníme tak, že začneme počátečním neterminálem S_0 vpravo nahoře a jednotlivé neterminály budeme v obrázku rozvíjet, jak udává tabulka 3.1. Získáme tak obrázky podobné obrázkům stromů výpočtů mon-red-automatu M. Přitom budeme dbát na to, aby se hrany stromu nekřížily.

Gramatikou mon-red-automatu M je gramatika G_M, kterou získáme redukcí

gramatiky[1]

$$G = \big(V, N, S, P\big),$$

kde

- $V = \Sigma_M \cup \{«, »\}$ je množina terminálů;
- N je množina neterminálů obsahující všechny pětice (a, s, u, o, v) tvořené
  - symbolem a abecedy Σ_M automatu M nebo levým resp. pravým omezovačem « resp. »,
  - koncovým nebo nekoncovým stavem s automatu M,
  - slovem u délky nejvýše k_M[2] tvořeným symboly abecedy Σ_M případně začínajícím levým omezovačem « nebo končícím pravým omezovačem »,
  - slovem v délky nejvýše k_M tvořeným symboly abecedy Σ_M případně končícím pravým omezovačem » a
  - operací o, která je rovna $\mathtt{ACC}$ nebo $\mathtt{RED}(n)$ pro libovolnou redukční posloupnost n takovou, že $\mathtt{RED}(nn') \in F_M$ pro nějaké n';
- $S = («, s_M, \lambda, \mathtt{ACC}, \lambda) \in N$ je počáteční neterminál, kde s_M je počáteční stav automatu M;
- P je množina pravidel definovaných pro libovolné neterminály $X, Y, Z \in N$, terminály $a, b \in V$, nekoncový stav s automatu M, slovo u délky nejvýše k_M tvořené symboly abecedy Σ_M případně začínající levým omezovačem « nebo končící pravým omezovačem », slovo v délky nejvýše k_M tvořené symboly abecedy Σ_M případně končící pravým omezovačem », redukční posloupnost n a $i \in \{\mathtt{0}, \mathtt{1}\}$ takto:

$$\begin{array}{lll}
X \to aY, & \text{je-li} & X = (a, s, \lambda, \mathtt{RED}(n), (b/i) \cdot v) \quad \text{a} \\
 & & Y = (b, f_M(s, b), \lambda, \mathtt{RED}(n \cdot i), v) \\
 & \text{nebo} & X = (a, s, \lambda, \mathtt{ACC}, \lambda) \quad \text{a} \\
 & & Y = (b, f_M(s, b), \lambda, \mathtt{ACC}, \lambda), \\
 & & \\
X \to Y, & \text{je-li} & X = (a, s, au, \mathtt{RED}(n), (b/i) \cdot v) \quad \text{a} \\
 & & Y = (b, f_M(s, b), u, \mathtt{RED}(n \cdot i), v) \\
 & \text{nebo} & X = (a, s, au, \mathtt{ACC}, \lambda) \quad \text{a} \\
 & & Y = (b, f_M(s, b), u, \mathtt{ACC}, \lambda),
\end{array}$$

[1] Redukcí gramatiky myslíme omezení množiny neterminálů jen na ty, které

- jsou dosažitelné z počátečního neterminálu S_0 a
- lze z nich vygenerovat nějaké terminálové slovo,

a omezení množiny pravidel jen na pravidla obsahující pouze neterminály z nové množiny neterminálů.

[2] Konstantu k_M jsme definovali na straně 10 jako délku nejdelší redukční posloupnosti automatu M.

$$
\begin{aligned}
S_0 &= (\text{«}, s_0, \lambda, \texttt{ACC}, \lambda) & S_1 &= (\text{»}, \texttt{ACC}, \lambda, \texttt{ACC}, \lambda)\\
S_2 &= (\text{»}, \texttt{ACC}, \text{»}, \texttt{ACC}, \lambda) & S_3 &= (\texttt{)}, \texttt{RED(101)}, \lambda, \texttt{RED(101)}, \lambda)\\
S_4 &= (\texttt{)}, \texttt{RED(101)}, \texttt{)}, \texttt{RED(101)}, \lambda) & S_5 &= (\texttt{)}, \texttt{RED(110)}, \lambda, \texttt{RED(110)}, \lambda)\\
S_6 &= (\texttt{)}, \texttt{RED(110)}, \texttt{)}, \texttt{RED(110)}, \lambda) & S_7 &= (\text{»}, \texttt{RED(110)}, \lambda, \texttt{RED(110)}, \lambda)\\
S_8 &= (\text{»}, \texttt{RED(110)}, \text{»}, \texttt{RED(110)}, \lambda) & S_9 &= (\text{«}, s_0, \text{«}\texttt{a}, \texttt{ACC}, \lambda)\\
S_{10} &= (\texttt{a}, s_1, \lambda, \texttt{ACC}, \lambda) & S_{11} &= (\texttt{a}, s_1, \texttt{a}, \texttt{ACC}, \lambda)\\
S_{12} &= (\texttt{a}, s_1, \texttt{a}\text{»}, \texttt{ACC}, \lambda) & S_{13} &= (\texttt{+}, s_2, \lambda, \texttt{RED(1)}, \texttt{)})\\
S_{14} &= (\texttt{+}, s_2, \lambda, \texttt{RED(1)}, \text{»}) & S_{15} &= (\texttt{+}, s_2, \texttt{+a}, \texttt{RED(1)}, \texttt{)})\\
S_{16} &= (\texttt{+}, s_2, \texttt{+a}, \texttt{RED(1)}, \text{»}) & S_{17} &= (\texttt{a}, s_3, \lambda, \texttt{RED(11)}, \texttt{)})\\
S_{18} &= (\texttt{a}, s_3, \lambda, \texttt{RED(11)}, \text{»}) & S_{19} &= (\texttt{a}, s_3, \texttt{a}, \texttt{RED(11)}, \texttt{)})\\
S_{20} &= (\texttt{a}, s_3, \texttt{a}, \texttt{RED(11)}, \text{»}) & S_{21} &= (\texttt{a}, s_3, \texttt{a)}, \texttt{RED(11)}, \texttt{)})\\
S_{22} &= (\texttt{a}, s_3, \texttt{a}\text{»}, \texttt{RED(11)}, \text{»}) & S_{23} &= (\texttt{(}, s_4, \lambda, \texttt{RED(1)}, \texttt{a})\\
S_{24} &= (\texttt{(}, s_4, \texttt{(a}, \texttt{RED(1)}, \texttt{a}) & S_{25} &= (\texttt{a}, s_5, \lambda, \texttt{RED(10)}, \lambda)\\
S_{26} &= (\texttt{a}, s_5, \texttt{a}, \texttt{RED(10)}, \lambda) & S_{27} &= (\texttt{a}, s_5, \texttt{a)}, \texttt{RED(10)}, \lambda)
\end{aligned}
$$

Obrázek 3.1: Neterminály gramatiky z příkladu 3.1.1.

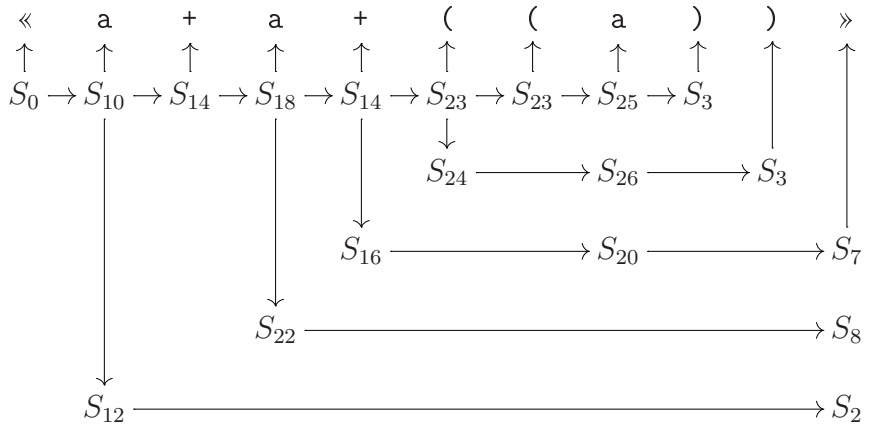
$$
\begin{aligned}
X \to aYZ, \quad & \text{je-li} \quad X = (a, s, \lambda, o, v) \quad \text{a}\\
& \qquad\quad\;\; Y = (b, f_M(s, b), \lambda, \texttt{RED(1)}, x) \quad \text{a}\\
& \qquad\quad\;\; Z = (a, s, ax, o, v),\\[1em]
X \to YZ, \quad & \text{je-li} \quad X = (a, s, au, o, v) \quad \text{a}\\
& \qquad\quad\;\; Y = (b, f_M(s, b), u, \texttt{RED(1)}, x) \quad \text{a}\\
& \qquad\quad\;\; Z = (a, s, ax, o, v),\\[1em]
X \to a, \quad & \text{je-li} \quad X = (a, \texttt{RED}(n), \lambda, \texttt{RED}(n), \lambda)\\
& \text{nebo} \quad X = (a, \texttt{ACC}, \lambda, \texttt{ACC}, \lambda),\\[1em]
X \to \lambda, \quad & \text{je-li} \quad X = (a, \texttt{RED}(n), a, \texttt{RED}(n), \lambda)\\
& \text{nebo} \quad X = (a, \texttt{ACC}, a, \texttt{ACC}, \lambda).
\end{aligned}
$$

G a tedy i G_M je zřejmě bezkontextovou gramatikou. Pro pochopení záměru konstrukce gramatiky doporučujeme čtenáři projít následující příklad, pomocí něhož základní myšlenky konstrukce v následujícím textu vysvětlujeme.

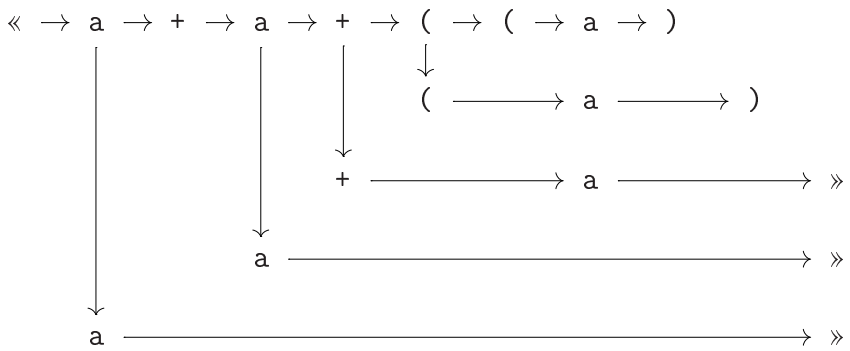
Příklad 3.1.1. Gramatika G_M mon-red-automatu M rozpoznávajícího jazyk zjednodušených aritmetických výrazů má neterminály na obrázku 3.1 a pravidla na obrázku 3.2. Derivační strom, který dává slovo «`a+a+((a))`», nakreslený naším způsobem je na obrázku 3.3a. Povšimněme si, že mezi jeho maximálními neterminálovými cestami a etapami výpočtu automatu M nad stejným slovem je

$$
\begin{array}{ll}
S_0 \to \text{«}\, S_{10} \mid \text{«}\, S_{23}\, S_9 & S_1 \to \text{»} \\
S_2 \to \lambda & S_3 \to \texttt{)} \\
S_4 \to \lambda & S_5 \to \texttt{)} \\
S_6 \to \lambda & S_7 \to \text{»} \\
S_8 \to \lambda & S_9 \to S_{11} \\
S_{10} \to \texttt{a}\, S_1 \mid \texttt{a}\, S_{14}\, S_{12} & S_{11} \to S_1 \mid S_{14}\, S_{12} \\
S_{12} \to S_2 & S_{13} \to \texttt{+}\, S_{17} \mid \texttt{+}\, S_{23}\, S_{15} \\
S_{14} \to \texttt{+}\, S_{18} \mid \texttt{+}\, S_{23}\, S_{16} & S_{15} \to S_{19} \\
S_{16} \to S_{20} & S_{17} \to \texttt{a}\, S_5 \mid \texttt{a}\, S_{13}\, S_{21} \\
S_{18} \to \texttt{a}\, S_7 \mid \texttt{a}\, S_{14}\, S_{22} & S_{19} \to S_5 \mid S_{13}\, S_{21} \\
S_{20} \to S_7 \mid S_{14}\, S_{22} & S_{21} \to S_6 \\
S_{22} \to S_8 & S_{23} \to \texttt{(}\, S_{25} \mid \texttt{(}\, S_{23}\, S_{24} \\
S_{24} \to S_{26} & S_{25} \to \texttt{a}\, S_3 \mid \texttt{a}\, S_{13}\, S_{27} \\
S_{26} \to S_3 \mid S_{13}\, S_{27} & S_{27} \to S_4
\end{array}
$$

Obrázek 3.2: Pravidla gramatiky z příkladu 3.1.1.



(a) derivační strom



(b) jeho zúžení na terminály v prvních složkách neterminálů

Obrázek 3.3: Derivační strom, který dává slovo «a+a+((a))».

vzájemně jednoznačná korespondence. Ta je dána prvními dvěma složkami neterminálů a definicí pravidel gramatiky: Vede-li z neterminálu S_1 vodorovná hrana do neterminálu S_2, pak je S_2 prvním neterminálem na pravé straně nějakého pravidla pro přepis neterminálu S_1 a pro nějaké symboly a_1, a_2 a stavy s_1, s_2 je

$$(a_1, s_1, \ldots) = S_1 \rightarrow S_2 = (a_2, s_2, \ldots) \qquad f_M(s_1, a_2) = s_2$$

Vede-li z neterminálu S_1 svislá hrana do neterminálu S_2, pak je S_2 druhým neterminálem na pravé straně nějakého pravidla pro přepis neterminálu S_1 a pro nějaký symbol a a stav s je

$$\begin{array}{l} S_1 = (a, s, \ldots) \\ \downarrow \\ S_2 = (a, s, \ldots) \end{array}$$

Pokud obrázek 3.3a zúžíme jen na první složky neterminálů, získáme strom výpočtu na obrázku 3.3b. V tomto stromu obsahuje každá cesta z kořene do listu prefix pracovního seznamu navštívený automatem M v příslušné etapě výpočtu automatu M nad slovem `a+a+((a))`. Vidíme, že cesty z kořene do listu, které ve stromu výpočtu odpovídají etapám výpočtu automatu nad daným slovem, jednoznačně korespondují s maximálními neterminálovými cestami (cestami z kořene do neterminálového listu) derivačního stromu, který dává stejné slovo.

Následnost etap je dána třetí, čtvrtou a pátou složkou neterminálů:

- Třetí složka neterminálu určuje, zda byl symbol v jeho první složce již vygenerován na některé z předchozích maximálních neterminálových cest derivačního stromu (resp. navštíven pracovní hlavou automatu v některé z předchozích etap).

- Čtvrtá složka složka říká, kde před neterminálem se odpojuje následující maximální neterminálová cesta.

- Poslední, pátá složka určuje symboly v prvních složkách neterminálů následující cesty za rozvětvením.

Derivační stromy a etapy výpočtů mon-red-automatu. Nyní si ukážeme, jak od derivačního stromu, který dává slovo w redukovatelné automatem M na slovo w', přejít k derivačnímu stromu, který dává slovo w'. Nejprve zavedeme následující značení: Je-li

$$X = (a, s, u, o, v)$$

libovolný neterminál gramatiky G_M, pak

$$X^\lambda = (a, s, \lambda, o, v).$$

X^λ získáme z neterminálu X nahrazením slova u ve třetí složce prázdným slovem λ. Přímo z definice pravidel gramatiky G_M plyne platnost následujících implikací, kde a je symbol v první složce pětice X, ω je libovolný neterminál gramatiky G_M

$$
\begin{tikzcd}
« & a_1 & \dots & a_{i-1} & a_i & a_{i+1} & \dots & a_{m_1} & a'_j & \dots & a'_{m_2} \\
S_0 \arrow[u] \arrow[r] & S_1 \arrow[u] \arrow[r] & \dots \arrow[r] & S_{i-1} \arrow[u] \arrow[r] & S_i \arrow[u] \arrow[d] \arrow[r] & S_{i+1} \arrow[u] \arrow[r] & \dots \arrow[r] & S_{m_1} \arrow[u] & & & \\
 & & & & S'_i \arrow[rr] & & \dots \arrow[rr] & & S'_j \arrow[uu] \arrow[r] & \dots \arrow[r] & S'_{m_2} \arrow[uu]
\end{tikzcd}
$$

(a) první dvě maximální neterminálové cesty původního stromu

$$
\begin{tikzcd}
« & a_1 & \dots & a_{i-1} & a'_i & \dots & a'_j & \dots & a'_{m_2} \\
S_0 \arrow[u] \arrow[r] & S_1 \arrow[u] \arrow[r] & \dots \arrow[r] & S_{i-1} \arrow[u] \arrow[r] & S'^{\lambda}_i \arrow[u] \arrow[rr] & \dots \arrow[rr] & S'^{\lambda}_j \arrow[u] \arrow[r] & \dots \arrow[r] & S'^{\lambda}_{m_2} \arrow[u]
\end{tikzcd}
$$

(b) první maximální neterminálová cesta výsledného stromu

Obrázek 3.4: Funkce R.

nebo prázdné slovo:

$$X \in V \implies X^\lambda \in V \tag{3.1}$$

$$X \to aYZ \in P \implies X = Z^\lambda \tag{3.2}$$

$$X \to Y\omega \in P \implies X^\lambda \to aY^\lambda\omega \in P \tag{3.3}$$

$$X \to \lambda \in P \implies X^\lambda \to a \in P \tag{3.4}$$

$$X \to aY\omega \in P \implies X = X^\lambda \text{ a } Y = Y^\lambda \tag{3.5}$$

Dále zavedeme funkci R, která libovolnému derivačnímu stromu $\mathcal{T}$ gramatiky G_M s alespoň dvěma maximálními neterminálovými cestami přiřadí strom $R(\mathcal{T})$. Ukážeme, že výsledný strom je rovněž derivačním stromem gramatiky G_M, a že má některé další, pro naše účely výhodné vlastnosti. Nejprve ale popíšeme jeho konstrukci.

Předpokládejme, že první dvě maximální neterminálové cesty stromu $\mathcal{T}$ spolu s terminály, které generují, jsou nakreslené na obrázku 3.4a. Z libovolného z vrcholů $S_0, \dots, S_{i-1}, S'_i, \dots, S'_{m_2}$ může (ale nemusí) vést další svislá hrana dolů do nějakého dalšího neterminálu. Z vrcholu S_i vede svislá hrana dolů do vrcholu S'_i, z vrcholů $S_{i+1}, \dots, S_{m_1}$ žádná svislá hrana směrem dolů nevede.

Strom $R(\mathcal{T})$ sestrojíme následujícím způsobem:

1. Odstraníme terminální vrcholy $a_i, \dots, a_{m_1}$ a $a'_j, \dots, a'_{m_2}$.
2. Odstraníme vrcholy $S_i, \dots, S_{m_1}$.
3. Pro každé $l \in \{i, \dots, m_2\}$ provedeme následující:
   (a) vrchol S'_l nahradíme vrcholem S'^{λ}_l
   (b) přidáme terminální vrchol a'_l, kde a'_l je symbol v první složce neterminálu S'^{λ}_l
   (c) přidáme hranu z S'^{λ}_l do a'_l

4. Přidáme vodorovnou hranu z vrcholu S_{i-1} do vrcholu $S_l'^{\lambda}$.

Z prvních dvou maximálních neterminálových cest stromu $\mathcal{T}$ (včetně terminálů, které generují) na obrázku 3.4a dostaneme funkcí R cestu nakreslenou na obrázku 3.4b.

Tvrzení 3.1.2. *Pro každý derivační strom $\mathcal{T}$ gramatiky G_M, který leží v definičním oboru funkce R (obsahuje alespoň dvě maximální neterminálové cesty), platí následující tvrzení:*

(i) $R(\mathcal{T})$ je derivačním stromem gramatiky G_M

(ii) $R(\mathcal{T})$ obsahuje o jednu maximální neterminálovou cestu méně než $\mathcal{T}$

(iii) dává-li $\mathcal{T}$ slovo w a $R(\mathcal{T})$ slovo w', pak $w \Rightarrow_M w'$

Důkaz. Předpokládejme, že první dvě maximální neterminálové cesty stromu $\mathcal{T}$ spolu s generovanými terminály vypadají stejně jako na obrázku 3.4a.

Tvrzení (i). Z (3.1) plyne, že vrcholy $S_i'^{\lambda}, \ldots, S_j'^{\lambda}, \ldots, S_{m_2}'^{\lambda}$ na obrázku 3.4b jsou neterminály gramatiky G_M.

V první maximální neterminálové cestě stromu $\mathcal{T}$ jsou použitá pravidla

$$\begin{aligned} S_{i-1} &\to a_{i-1}\, S_i\, \omega \\ S_i &\to a_i\, S_{i+1}\, S_i' \end{aligned}$$

pro nějaké $\omega \in V \cup \{\lambda\}$. Podle (3.2) je $S_i = S_i'^{\lambda}$, takže

$$S_{i-1} \to a_{i-1}\, S_i'^{\lambda}\, \omega$$

je pravidlo gramatiky G_M. Protože v druhé maximální neterminálové cestě stromu $\mathcal{T}$ jsou použitá pravidla

$$\begin{aligned} S_i' &\to S_{i+1}'\, \omega_{i+1} \\ &\vdots \\ S_{j-1}' &\to S_j'\, \omega_j \\ S_j' &\to a_j'\, S_{j+1}'\, \omega_{j+1} \\ &\vdots \\ S_{m_2-1}' &\to a_{m_2-1}'\, S_{m_2}'\, \omega_{m_2} \\ S_{m_2}' &\to a_{m_2}' \end{aligned}$$

pro nějaká $\omega_{i+1}, \ldots, \omega_{m_2} \in V \cup \{\lambda\}$, jsou podle (3.3), (3.4) a (3.5)

$$\begin{aligned} S_i'^{\lambda} &\to a_i'\, S_{i+1}'^{\lambda}\, \omega_{i+1} \\ &\vdots \\ S_{j-1}'^{\lambda} &\to a_{j-1}'\, S_j'^{\lambda}\, \omega_j \\ S_j'^{\lambda} &\to a_j'\, S_{j+1}'^{\lambda}\, \omega_{j+1} \\ &\vdots \\ S_{m_2-1}'^{\lambda} &\to a_{m_2-1}'\, S_{m_2}'^{\lambda}\, \omega_{m_2} \\ S_{m_2}'^{\lambda} &\to a_{m_2}' \end{aligned}$$

pravidla gramatiky G_M. Strom $R(\mathcal{T})$ je tedy derivačním stromem gramatiky G_M.

Tvrzení (ii). Při konstrukci stromu $R(\mathcal{T})$ ze stromu $\mathcal{T}$ došlo k odstranění sufixu první maximální neterminálové cesty hned za posledním jejím „neterminálovým rozvětvením“. Všechny ostatní neterminály byly případně nahrazeny jinými neterminály a všechny ostatní hrany mezi neterminály ve stromu zůstaly. Derivační strom $R(\mathcal{T})$ tedy obsahuje o jednu maximální neterminálovou cestu méně než derivační strom $\mathcal{T}$.

Tvrzení (iii). Dává-li strom $\mathcal{T}$ slovo w a jeho první dvě maximální neterminálové cesty jsou na obrázku 3.4a, pak zřejmě

$$w = a_0 a_1 \dots a_i a_{i+1} \dots a_{m_1} x$$

pro nějaké x. Neterminál S_i je posledním společným netermínálem prvních dvou maximálních neterminálových cest stromu $\mathcal{T}$ a je přepsán použitím pravidla gramatiky G_M na $a_i\, S_{i+1}\, S'_i$. Z definice pravidel gramatiky G_M pro každé $l \in \{i, \dots, m_1 - 1\}$ plyne, že $s_{l+1} = f_M(s_l, a_{l+1})$, kde s_l je stav obsažený ve druhé složce neterminálu S_l, s_{m_1} je stav obsažený ve druhé složce neterminálu S_{m_1}, $s_{m_1} = \mathtt{RED}(n)$ pro nějakou redukční posloupnost n,

$$\begin{array}{rcccl} & & a_i & & \\ & & \uparrow & & \\ (a_i, s_i, \lambda, o, v) & = & S_i \to S_{i+1} & = & (a_{i+1}, s_{i+1}, \lambda, \mathtt{RED}(1), u) \\ & & \downarrow & & \\ (a_i, s_i, a_i \cdot u, o, v) & = & S'_i & & \end{array}$$

pro nějaká slova u a v a nějakou operaci o a $u = (a_{i+1} \dots a_{m_1})/n = a'_{i+1} \dots a'_{j-1}$. Dohromady dostáváme, že $w' = a_0 a_1 \dots a_i u x$, a tedy $w \Rightarrow_M w'$. □

Tvrzení 3.1.3. *Pro libovolný derivační strom $\mathcal{T}$ gramatiky G_M a libovolné slovo «w'» platí: Dává-li $\mathcal{T}$ slovo «w» a «w'» $\Rightarrow_M$ «w», pak existuje derivační strom $\mathcal{T}'$ gramatiky G_M, který dává slovo «w'», a $R(\mathcal{T}') = \mathcal{T}$.*

Důkaz. Jestliže «w'» $\Rightarrow_M$ «w», pak pro nějaká u, v je uv prefix slova w'», $\delta^*_M(s_M, uv) = \mathtt{RED}(n)$ a $|v| = |n|$. Nechť $S_0 \to S_1 \to \dots \to S_m$ je první maximální neterminálová cesta derivačního stromu $\mathcal{T}$ a první složky neterminálů na této cestě po řadě obsahují terminální symboly « $= a_0, a_1, \dots, a_m$. Protože je automat M monotónní, je $m \geq |uv/n| = l$ a $uv/n = a_1 \dots a_k \dots a_l$, kde $k = |u|$.

Strom $\mathcal{T}'$ získáme z stromu $\mathcal{T}$ následujícím způsobem:

1. Odstraníme terminální symboly $a_k, \dots, a_l$.
2. Neterminály $S_k, \dots, S_l$ na první maximální neterminálové cestě nahradíme neterminály $S'_k, \dots, S'_l$, které získáme z původních neterminálů záměnou prázdného slova λ ve třetí složce po řadě slovy

$$\begin{array}{r} a_k a_{k+1} \dots a_l, \\ a_{k+1} \dots a_l, \\ \vdots \\ a_l. \end{array}$$

3. Přidáme nové neterminály

$$S''_k = (a_k, s_k, \lambda, o, v) = S_k,$$
$$S''_{k+i} = (b_i, s_{k+i}, \lambda, \mathtt{RED}(n'_i), v_i/n_i),$$

kde b_i je i-tý symbol slova v zleva, v'_i je prefix slova v délky i, $v'_i v_i = v$, n'_i je prefix redukční posloupnosti délky i, $n'_i n_i = n$ a $s_{k+i} = \delta_M(s_k, v'_i)$ pro všechna $i \in \{1, \ldots, |v|\}$.

4. Přidáme terminál a_k a hranu z neterminálu S''_k do tohoto terminálu. Podobně přidáme i terminály $b_1, \ldots, b_{|v|}$ a připojíme je do nového derivačního stromu hranou vedoucí po řadě z neterminálu $S''_{k+1}, \ldots, S''_{k+|v|}$.

5. Neterminál S''_k připojíme do nového derivačního stromu hranou vedoucí z neterminálu S_{k-1}. Využijeme přitom pravidla gramatiky $S_{k-1} \to a_{k-1}\, S''_k S'_k$. Podobně do derivačního stromu připojíme i neterminály $S''_{k+1}, \ldots, S''_{k+|v|}$ pomocí pravidel $S''_k \to a_k\, S''_{k+1}, \ldots, S''_{k+|v|-1} \to a_{k+|v|-1}\, S''_{k+|v|}$.

Uvedenou konstrukcí jsme získali derivační strom $\mathcal{T}'$, který dává slovo «w'» a $R(\mathcal{T}') = \mathcal{T}$. Korektnost této konstrukce můžeme ověřit z definice neterminálů a pravidel gramatiky mon-red-automatu M. □

Tvrzení 3.1.4. *Nechť p je libovolné přirozené číslo, $\mathcal{T}_1, \ldots, \mathcal{T}_p$ jsou libovolné derivační stromy gramatiky G_M a $w_1, \ldots, w_p$ jsou libovolná slova ze «Σ^*_M». Pokud zároveň*

- *$\mathcal{T}_i = R(\mathcal{T}_{i+1})$ pro všechna $i \in \{1, \ldots, p-1\}$,*
- *$\mathcal{T}_i$ dává slovo w_i pro všechna $i \in \{1, \ldots, p\}$,*

pak zároveň

- *$w_{i+1} \Rightarrow_M w_i$ pro všechna $i \in \{1, \ldots, p-1\}$,*
- *w_1 je akceptované automatem M v jediné etapě.*

Důkaz. Větu dokážeme indukcí podle přirozeného čísla p. Zřejmě platí, že $\mathcal{T}_i$ obsahuje právě i maximálních neterminálových cest.

$p = 1$. Nechť $\mathcal{T}_1$ je derivační strom gramatiky G_M, který dává slovo w_1 a obsahuje právě jednu maximální neterminálovou cestu

$$S_0 \to S_1 \to \cdots \to S_{i-1} \to S_i \to \cdots \to S_m.$$

To znamená, že v tomto stromu není použité žádné pravidlo s dvěma neterminály na pravé straně. Ve stromu $\mathcal{T}_1$ jsou tedy použitá pouze pravidla

$$S_{i-1} \to a_i \mid a_i\, S_i \mid S_i \mid \lambda$$

Slovo ve třetí složce neterminálu S_i je (pro všechna $i \in \{1, \ldots, m\}$) prázdné, protože je prázdné v počátečním neterminálu S_0 a pro všechna pravidla $S_{i-1} \to$

$a_i\, S_i \mid S_i$ platí, že slovo ve třetí složce neterminálu S_i není delší než slovo ve třetí složce neterminálu S_{i-1}. Ve stromu $\mathcal{T}_1$ jsou tedy použitá jen pravidla

$$S_{i-1} \to a_i \mid a_i\, S_i$$

Strom výpočtu $\mathcal{T}_1$ pro slovo w_1 vypadá tudíž takto:

$$\begin{array}{ccccccccccc} « & & a_1 & & \dots & & a_{i-1} & & a_i & & \dots & & a_m \\ \uparrow & & \uparrow & & & & \uparrow & & \uparrow & & & & \uparrow \\ S_0 & \to & S_1 & \to & \dots & \to & S_{i-1} & \to & S_i & \to & \dots & \to & S_m \end{array}$$

Z definice pravidel gramatky G_M ihned plyne, že $f_M(s_{i-1}, a_i) = s_i$ pro všechna $i \in \{1, \dots, m\}$, kde s_0 je stav ve druhé složce počátečního neterminálu S_0 a zároveň počáteční stav automatu M, s_i je stav ve druhé složce neterminálu S_i.

Protože v pravidlech $S_{i-1} \to a_i\, S_i$ se shoduje typ operace (`ACC` nebo `RED`) ve čtvrté složce neterminálu S_{i-1} s typem operace ve čtvrté složce neterminálu S_i a v počátečním neterminálu S_0 je operace `ACC`, je operace `ACC` ve všech neterminálech stromu $\mathcal{T}_1$.

Automat M akceptuje až po přesunu přes pravý omezovač », takže $a_m =$ ». Dostáváme $w = a_0 a_1 \dots a_{m-1} a_m =$ «w'» a

$$\delta_M^*(s_M, w'») = \texttt{ACC}$$

Indukční krok. Předpokládejmě, že věta platí pro $p = q \geq 1$. Ukážeme, že platí i pro $p = q + 1$. Nechť $\mathcal{T}$ je libovolný derivační strom gramatiky G_M s $q + 1$ maximálními neterminálovými cestami, který dává slovo w. Pak podle 3.1.2 je $\mathcal{T}' = R(\mathcal{T})$ derivační strom gramatiky G_M s q maximálními neterminálovými cestami a dává-li slovo w', pak $w \Rightarrow_M w'$. Pro strom $\mathcal{T}'$ věta podle indukčního předpokladu platí, takže platí i pro strom $\mathcal{T}$. □

Tvrzení 3.1.5. *Nechť p je libovolné přirozené číslo, a $w_1, \dots, w_p$ jsou libovolná slova ze «Σ_M^*». Pokud zároveň*

- *$w_{i+1} \Rightarrow_M w_i$ pro všechna $i \in \{1, \dots, p-1\}$,*
- *w_1 je akceptované automatem M v jediné etapě.*

pak pro nějaké derivační stromy $\mathcal{T}_1, \dots, \mathcal{T}_p$ gramatiky G_M platí zároveň

- *$\mathcal{T}_i = R(\mathcal{T}_{i+1})$ pro všechna $i \in \{1, \dots, p-1\}$,*
- *$\mathcal{T}_i$ dává slovo w_i pro všechna $i \in \{1, \dots, p\}$.*

Důkaz. Větu dokážeme indukcí podle přirozeného čísla p.

$p = 1$. Předpokládejme, že slovo w je přijato automatem M v jediné etapě. Pro nějaká $s_0, \dots, s_m$ a $a_1, \dots, a_m$ je tedy

$$\begin{aligned} \delta_M(s_0, a_1) &= s_1 \\ &\vdots \\ \delta_M(s_{i-1}, a_i) &= s_i \\ &\vdots \\ \delta_M(s_{m-1}, a_m) &= s_m \end{aligned}$$

kde s_0 je počáteční stav stav automatu M, $s_m = \texttt{ACC}$ a $a_m = »$. Z toho plyne, že

$$\begin{aligned} S_0 &= (a_0, s_0, \lambda, \texttt{ACC}, \lambda) \\ S_1 &= (a_1, s_1, \lambda, \texttt{ACC}, \lambda) \\ &\vdots \\ S_i &= (a_i, s_i, \lambda, \texttt{ACC}, \lambda) \\ &\vdots \\ S_m &= (a_m, s_m, \lambda, \texttt{ACC}, \lambda) \end{aligned}$$

kde s_0 je počáteční stav automatu M, $s_m = \texttt{ACC}$, $a_0 = «$ a $a_m = »$ jsou neterminály gramatiky G_M. S_0 je zřejmě počáteční neterminál gramatiky G_M a

$$\begin{aligned} S_0 &\to a_0\, S_1 \\ &\vdots \\ S_{i-1} &\to a_{i-1}\, S_i \\ &\vdots \\ S_{m-1} &\to a_{m-1}\, S_m \\ S_m &\to a_m \end{aligned}$$

jsou pravidla gramatiky G_M. Můžeme tedy zkonstruovat následující derivační strom gramatiky G_M, který dává slovo $w = «a_1 \dots a_{m-1}»$:

$$\begin{array}{ccccccccccccc} « & & a_1 & & \dots & & a_{i-1} & & a_i & & \dots & & » \\ \uparrow & & \uparrow & & & & \uparrow & & \uparrow & & & & \uparrow \\ S_0 & \to & S_1 & \to & \cdots & \to & S_{i-1} & \to & S_i & \to & \cdots & \to & S_m \end{array}$$

Indukční krok. Předpokládejmě, že věta platí pro $p = q \geq 1$. Ukážeme, že platí i pro $p = q + 1$. Nechť w je libovolné slovo jazyka $« L(M) »$, které automat M přijímá v právě q etapách. Nechť dále w' je libovolné slovo takové, že $w' \Rightarrow_M w$. Podle indukčního předpokladu tedy nějaký derivační strom $\mathcal{T}$ gramatiky G_M dává slovo w. Podle 3.1.3 pak existuje derivační strom $\mathcal{T}'$, který dává slovo w' a $\mathcal{T} = R(\mathcal{T}')$. Věta tedy platí i pro slovo w', které automat M přijímá v $q + 1$ etapách. □

Přímým důsledkem tvrzení 3.1.4 a 3.1.5 je následující věta.

Věta 3.1.6. $L(G_M) = «L(M)»$.

Další věta říká, že gramatika G_M monotónního redukčního automatu M je vhodná pro konstrukci klasického syntaktického analyzátoru pro jazyk rozpoznávaný tímto automatem:

Věta 3.1.7. *G_M je* LR(0)*-gramatika.*

Důkaz. Nejprve si připomeňme několik klasických pojmů zaváděných v souvislosti s LR(0)-gramatikami, jak jsou uvedeny například v [5].

Položkou bezkontextové gramatiky G nazveme každý výraz typu

$$X \to \alpha . \beta,$$

kde $X \to \alpha\beta$ je pravidlo gramatiky G. Speciálně je $X \to \bullet$ položkou, pokud $X \to \lambda$ je pravidlem gramatiky G. Pro každé pravidlo $X \to \gamma$ nazýváme $X \to \gamma\bullet$ *úplnou položkou*. Řekneme, že $A \to \alpha\bullet\beta$ je *platnou položkou pro řetěz* ω, jestliže existuje pravá větná forma ξAu (u je řetěz terminálů) taková, že $\xi\alpha = \omega$. Pro množinu všech platných položek pro řetěz ω budeme užívat označení $I(\omega)$.

Bezkontextová gramatika $G = (V, N, S, P)$ se nazývá $\mathsf{LR}(0)$-*gramatika*, jestliže splňuje tyto podmínky:

1. Počáteční symbol S se nevyskytuje na pravé straně žádného pravidla.
2. Absence konfliktů `reduce`/`reduce`: Pro libovolný řetěz $\gamma \in (V \cup N)^*$ se v množině $I(\gamma)$ vyskytuje nejvýše jedna úplná položka.
3. Absence konfliktů `shift`/`reduce`: Vyskytuje-li se v $I(\gamma)$ úplná položka, pak už v $I(\gamma)$ není žádná položka s terminálem napravo od tečky.

První podmínka je splněna, protože počáteční neterminál S se v žádném pravidle gramatiky G_M nevyskytuje na pravé straně.

Zbývající dvě podmínky dokážeme pomocí (souvislých) charakteristik množin položek postupem převzatým z [5]. Charakteristiku definujeme postupně, nejprve pro terminální a neterminální symboly, pak pro řetězce terminálních a neterminálních symbolů, položky gramatiky a nakonec i pro množiny položek získané konstrukcí stavů položkového automatu gramatiky G_M. Ukážeme, že každá množina položek, která je stavem položkového automatu gramatiky G_M, má souvislou charakteristiku, což má za následek absenci konfliktů typu `reduce`/`reduce` i `shift`/`reduce` ve stavech položkového automatu gramatik G_M automatu M.

Pro každý terminální i neterminální symbol x gramatiky G_M definujeme *charakteristiku* $[x]$ *symbolu* x takto:

$$[x] = \begin{cases} \texttt{initial}, & \text{je-li } x = S, \\ \texttt{terminal}, & \text{je-li } x \in V \\ (a, s, u), & \text{je-li } S \neq x = (a, s, u, o, v) \in N \text{ pro nějaká } o \text{ a } v. \end{cases}$$

Charakteristiku řetězů α utvořených z terminálů i neterminálů gramatiky G_M definujeme předpisem

$$[\alpha] = \begin{cases} \texttt{empty}, & \text{je-li } \alpha = \lambda, \\ [x], & \text{je-li } x \in V \cup N \text{ a } \alpha = x\alpha' \text{ pro nějaké } \alpha'. \end{cases}$$

Charakteristiku položky $p = X \to \alpha\bullet\beta$ definujeme takto:

$$[p] = ([X], \alpha, [\beta])$$

Charakteristikou množiny položek I gramatiky G_M budeme nazývat množinu

$$[I] = \{[p] \mid p \in I\}$$

Ověřením pro všechny typy pravidel gramatiky G_M dostaneme, že první dvě složky charakteristiky položky jednoznačně určují její třetí složku. Přesněji, jsou-li $(c_{1,1}, \alpha_1, c_{1,2})$ a $(c_{2,1}, \alpha_2, c_{2,2})$ charakteristiky dvou položek gramatiky G_M, pak

$$(c_{1,1} = c_{2,1} \quad \text{a} \quad \alpha_1 = \alpha_2) \quad \Longrightarrow \quad c_{1,2} = c_{2,2}$$

Množina položek I má *souvislou charakteristiku*, jestliže

$$[I] = \{(c_0, \alpha, c_1), (c_1, \lambda, c_2), \ldots, (c_{n-1}, \lambda, c_n)\}$$

pro nějaká c_0, c_1, c_2, ..., c_{n-1}, c_n a α.

Ověřením pro všechny typy pravidel gramatiky G_M lze dokázat platnost následujících tvrzení, která říkají, že konstrukcí stavů položkového automatu gramatiky G_M vznikají jen množiny položek se souvislou charakteristikou.

1. $I(\lambda)$ má souvislou charakteristiku.
2. Jestliže I má souvislou charakteristiku, $p = A \to \alpha.B\beta \in I$ a $I_p = \{B \to .\gamma \mid B \to \gamma \in G_M\}$, pak $I \cup I_p$ má souvislou charakteristiku.
3. Jestliže I má souvislou charakteristiku, pak pro každý symbol x gramatiky G_M má množina $I_x = \{A \to \alpha x.\beta \mid A \to \alpha.x\beta \in I\}$ také souvislou charakteristiku.

Libovolný stav $I(\gamma)$ položkového automatu gramatiky G_M má tedy souvislou charakteristiku. Z definice množiny položek se souvislou charakteristikou vyplývá, že taková množina může obsahovat nejvýše jednu trojici, jejíž třetí složka je `empty` nebo `terminal`. Každá úplná položka $A \to \alpha.$ má charakteristiku $([A], \alpha, \texttt{empty})$, každá položka $A \to \alpha.\beta$, v níž je napravo od tečky terminál, má charakteristiku $([A], \alpha, \texttt{terminal})$. Proto každá množina položek se souvislou charakteristikou může zahrnovat nejvýše jednu úplnou položku a pokud ji zahrnuje, pak již neobsahuje žádnou položku s terminálem napravo od tečky.

Každá množina $I(\gamma)$ tedy splňuje druhou a třetí podmínku definice LR(0)-gramatiky a G_M je LR(0)-gramatika. □

3.2 Prefixově korektní mon-red-automat

Předpokládejme, že M je monotónní redukční automat. Říkáme, že M je *prefixově korektní*, jestliže pro libovolné slovo $u \in \Sigma_M^*$ a redukční posloupnost n platí následující implikace:

$$\begin{aligned}
\delta_M^*(s_M, u) \in S_M &\implies \exists v \in \Sigma_M^* : uv \in L(M) \\
\delta_M^*(s_M, u) = \texttt{RED}(n) &\implies \exists v \in \Sigma_M^* : uv \in L(M) \\
\delta_M^*(s_M, u\text{»}) = \texttt{RED}(n) &\implies u \in L(M)
\end{aligned}$$

První dvě implikace říkají, že libovolné slovo u nad abecedou Σ_M, přes které automat M přesune svou pracovní hlavu a přitom převede svou řídící jednotku z počátečního stavu s_M do libovolného nekoncového nebo redukčního stavu, je prefixem nějakého slova jazyka $L(M)$.

Třetí implikace tvrdí, že libovolné slovo u nad abecedou Σ_M, přes které automat M přesune svou pracovní hlavu z počátečního stavu s_M, za ním ještě překročí pravý omezovač » a přitom převede svou řídící jednotku do libovolného redukčního stavu, je již jistě slovem jazyka $L(M)$. Přitom k formálnímu přijetí slova u může automat potřebovat ještě několik dalších etap.

Věta 3.2.1. *Ke každému mon-red-automatu lze zkonstruovat ekvivalentní prefixově korektní mon-red-automat.*

Důkaz. Předpokládejme, že M je mon-red-automat. K automatu M nejprve sestrojíme $\mathsf{LR}(0)$-gramatiku G_M. Tu využijeme při konstrukci redukčního automatu M'. Nakonec ukážeme, že M' je monotónní, prefixově korektní a ekvivalentní s původním redukčním automatem M.

Konstrukce redukčního automatu M'. Konstrukce redukčního automatu M' připomíná konstrukci položkového automatu pro gramatiku G_M. Nekoncovými stavy automatu M' budou, stejně jako u položkového automatu, množiny položek této gramatiky. Nad libovolným slovem nás ale budou zajímat jen přechody, které by u položkového automatu gramatiky G_M vedly do prvního stavu,

- který obsahuje úplnou položku nebo
- ze kterého nelze přejít přes snímaný terminál do dalšího stavu.

Tyto přechody odpovídají přesunu přes terminály generované první maximální neterminálovou cestou derivačního stromu gramatiky $G_{M'}$.

Z definice gramatiky G_M plyne, že obsahuje jen pravidla následujících typů

$$\begin{array}{ll} X \to a\,Y\,Z & X \to Y\,Z \\ X \to a\,Y & X \to Y \\ X \to a & X \to \lambda \end{array}$$

a na první maximální neterminálové cestě libovolného derivačního stromu gramatiky G_M jsou použítá jen pravidla typů uvedených v prvním sloupci, tedy s pravou stranou začínající terminálem. Poslední pravidlo použité na první maximální neterminálové cestě libovolného derivačního stromu gramatiky G_M je tedy typu $X \to a$ a první úplná položka, na kterou položkový automat narazí, je tím pádem typu $X \to a\,.$. Proto budeme stavy redukčního automatu M' konstruovat jen z následujících typů položek

$$\begin{array}{ll} X \to .\,a\,Y\,Z & X \to a\,.\,Y\,Z \\ X \to .\,a\,Y & X \to a\,.\,Y \\ X \to .\,a & X \to a\,. \end{array}$$

a přechody mezi množinami položek omezíme jen na přechody přes terminální symboly.

Nejprve pro libovolná $u \in \Sigma_M^*$ a $a \in \Sigma_M \cup \{«, »\}$ zkonstruujeme množiny položek

$$\begin{aligned} I(\lambda) &= \{S \to .\,«\,\gamma \mid S \to «\,\gamma \in P\}, \\ I(ua) &= \{X \to a\,.\,\alpha \mid X \to .\,a\,\alpha \in I(u)\} \\ &\quad \cup \{Y \to .\,b\,\beta \mid X \to .\,a\,Y\,\gamma \in I(u) \text{ a } Y \to b\,\beta \in P\}, \end{aligned}$$

kde S je počáteční neterminál gramatiky G_M. Uvedené množiny položek získáme stejným postupem, jakým se v klasické teorii k libovolné $\mathsf{LR}(0)$-gramatice konstruuje položkový automat. Uvažujeme ale jen položky výše uvedených typů. Naše množiny položek jsou tedy podmnožinami množin položek, které jsou stavy klasického položkového automatu, a tudíž neobsahují žádný konflikt *shift/reduce* ani

reduce/reduce. Nekoncovými stavy redukčního automatu M' jsou všechny zkonstruované množiny položek kromě $I(\lambda)$ a množin obsahujících pouze úplnou položku. Jeho počátečním stavem je množina $I(«)$. Redukčními koncovými stavy jsou všechny $\mathtt{RED}(n)$, pro které existuje úplná položka $X \to a\,.$ gramatiky $G(M')$ a $X = (a, \mathtt{RED}(n), \lambda, \mathtt{RED}(n), \lambda)$. Přechodovou funkci definujeme takto:

$$f_{M'}\big(I(u), a\big) = \begin{cases} \mathtt{RED}(n), & \text{je-li } I(ua) = \{X \to a\,.\} \text{ a} \\ & X = (a, \mathtt{RED}(n), \lambda, \mathtt{RED}(n), \lambda) \text{ pro nějaké } n, \\ \mathtt{ACC}, & \text{je-li } a = », I(u») = \{X \to »\,.\} \text{ a} \\ & X = (», \mathtt{ACC}, \lambda, \mathtt{ACC}, \lambda), \\ \mathtt{ERR}, & \text{je-li } I(ua) = \emptyset, \\ I(ua), & \text{jinak.} \end{cases}$$

Ekvivalence automatů M' a M. Zřejmě je $«L(M)» = L(G_M)$. Ukážeme, že

$$L(G_M) = «L(M')».$$

Předpokládejme, že $«w» \in L(G_M)$ a $\mathcal{T}$ je derivační strom gramatiky G_M, který dává $«w»$. Indukcí podle počtu maximálních neterminálových cest ve stromu $\mathcal{T}$ dokážeme, že $w \in L(M')$.

Pokud strom $\mathcal{T}$ obsahuje pouze jednu maximální neterminálovou cestu, pak $I(«w») = \{X \to »\,.\}$ a $X = (», \mathtt{ACC}, \lambda, \mathtt{ACC}, \lambda)$, takže $\delta^*_{M'}(I(«), w») = \mathtt{ACC}$ a tedy $w \in L(M')$.

Předpokládejme, že tvrzení platí pro všechny derivační stromy gramatiky G_M s nejvýše p maximálními neterminálovými cestami. Dokážeme, že pak platí i pro derivační stromy s $p+1$ maximálními neterminálovými cestami. Je-li $\mathcal{T}$ derivační strom s $p+1$ maximálními neterminálovými cestami, který dává slovo $«w»$, pak podle tvrzení 3.1.2 je $R(\mathcal{T})$ strom s p maximálními neterminálovými cestami, který dává slovo $«w'»$ takové, že $«w» \Rightarrow_M «w'»$. Podle indukčního předpokladu $w' \in L(M')$. Pro nějaký prefix $«ua$ slova $«w»$ je tedy $\delta^*_M(s_M, ua) = \mathtt{RED}(n)$, kde $I(«ua) = \{X \to a\,.\}$ a $X = (a, \mathtt{RED}(n), \lambda, \mathtt{RED}(n), \lambda)$, takže $\delta^*_{M'}(I(«u), a) = \mathtt{RED}(n)$, $«w» \Rightarrow_{M'} «w'»$ a tedy $w \in L(M')$.

Obrácenou inkluzi ($«L(M')» \subseteq L(G_M)$) dokážeme podobným způsobem indukcí podle počtu etap redukční analýzy slova $«w»$ automatem M' s využitím tvrzení 3.1.3.

Prefixová korektnost. Předpokládejme nejprve, že se automat M' přesune z počátečního stavu $I(«)$ přes prefix $u \in \Sigma^*_M$ slova $w»$ do stavu $I(«u)$, formálně

$$\delta^*_{M'}(I(«), u) = I(«u).$$

To znamená, že z množin $I(«)$, …, $I(«u)$ lze po řadě vybrat položky uvedené níže v levém sloupci a použít je v derivaci podle gramatiky G_M uvedené v pravém

sloupci

$$
\begin{array}{ll}
S \to \text{«} \,.\, S_1\,\gamma_1 & S \Rightarrow \text{«}\, S_1\,\gamma_1 \\
S_1 \to a_1 \,.\, S_2\,\gamma_2 & \quad \Rightarrow \text{«}\, a_1\, S_2\,\gamma_2\gamma_1 \\
\quad\vdots & \quad\vdots \\
S_k \to a_k \,.\, S_{k+1}\,\gamma_{k+1} & \quad \Rightarrow \text{«}\, \underbrace{a_1 \ldots a_k}_{u}\, S_{k+1}\gamma_{k+1}\, \underbrace{\gamma_k \ldots \gamma_2\,\gamma_1}_{\gamma},
\end{array}
$$

kde $a_1 \ldots a_k = u$. Protože je gramatika G_M redukovaná, lze všechny neterminály v $S_{k+1}\,\gamma_{k+1}\,\gamma$ přepsat na nějaký terminální řetězce, takže u je prefixem nějakého slova z $L(M)$.

Další možností je, že $\delta^*_{M'}(I(\text{«}), u) = \texttt{RED}(n)$ pro nějakou redukční posloupnost n. Potom $I(\text{«}u) = \{X \to a_k\,.\}$, kde $X = (a_k, \texttt{RED}(n), \lambda, \texttt{RED}(n), \lambda)$. Poslední položku $S_k \to a_k \,.\, S_{k+1}\,\gamma_{k+1}$ v seznamu výše stačí nahradit úplnou položkou $S_k \to a_k\,.$ a dostaneme derivaci $S \Rightarrow^*_{G_M} \text{«}\, u\,\gamma$. Je-li $a_k \neq \text{»}$, pak u je prefixem nějakého slova jazyka $L(M)$. Je-li $a_k = \text{»}$, pak $\gamma \Rightarrow^*_{G_M} \lambda$ (jinak by gramatika G_M generovala slovo mimo «Σ^*_M») a $u \in L(M)$».

Monotonie. Předpokládejme, že automat M' se přesune ze stavu $I(\text{«})$ přes prefix uv slova w» do stavu $\texttt{RED}(n) \in F_{M'}$, $|v| = |n|$, a že slovo u se skládá ze symbolů $a_1, \ldots, a_k \in \Sigma_M$ a slovo v ze symbolů $a_{k+1}, \ldots, a_{l-1} \in \Sigma_M$, $a_l \in \Sigma_M \cup \{\text{»}\}$. Stejně jako v důkazu prefixové korektnosti i nyní můžeme z množin $I(\text{«})$, …, $I(\text{«}uv)$ vybrat položky uvedené ve sloupci vlevo a jejich pravidla pak použít v odvození uvedené ve sloupci vpravo:

$$
\begin{array}{ll}
S \to \text{«} \,.\, S_1\,\gamma_1 & S \Rightarrow \text{«}\, S_1\,\gamma_1 \\
S_1 \to a_1 \,.\, S_2\,\gamma_2 & \quad \Rightarrow \text{«}\, a_1\, S_2\,\gamma_2\,\gamma_1 \\
\quad\vdots & \quad\vdots \\
S_{k-1} \to a_{k-1} \,.\, S_k\,\gamma_k & \quad \Rightarrow \text{«}\, a_1 \ldots a_{k-1}\, S_k\,\gamma_k \ldots \gamma_2\,\gamma_1 \\
S_k \to a_k \,.\, S_{k+1}\,X_1 & \quad \Rightarrow \text{«}\, \underbrace{a_1 \ldots a_{k-1}\, a_k}_{u}\, S_{k+1}\,X_0\, \underbrace{\gamma_k \ldots \gamma_2\,\gamma_1}_{\gamma} \\
S_{k+1} \to a_{k+1} \,.\, S_{k+2} & \quad \Rightarrow \text{«}\, u\, a_{k+1}\, S_{k+2}\,X_0\,\gamma \\
\quad\vdots & \quad\vdots \\
S_l \to a_l\,. & \quad \Rightarrow \text{«}\, u\, \underbrace{a_{k+1} \ldots a_{l-1}\, a_l}_{v}\, X_0\,\gamma = \text{«}\, uv\, X_0\,\gamma
\end{array}
$$

Protože G_M je redukovaná, lze z X_0 odvodit nějaký terminální řetězec. Předpokládejme, že $v/n = b_1 \ldots b_m$. Z definice gramatiky G_M plyne, že nějaké její neterminály $X_1, X_2, \ldots, X_m$ obsahují ve své první složce po řadě symboly b_1, b_2,

$\ldots, b_m$ a gramatika obsahuje následující pravidla

$$
\begin{array}{ll}
S_k \to a_k\, S_{k+1}\, X_0 & S_k = X_0^\lambda \\
X_0 \to X_1\, \gamma_1' & X_0^\lambda \to a_k\, X_1^\lambda\, \gamma_1' \\
X_1 \to X_2\, \gamma_2' & X_1^\lambda \to b_1\, X_2^\lambda\, \gamma_2' \\
X_2 \to X_3\, \gamma_3' & X_2^\lambda \to b_2\, X_3^\lambda\, \gamma_3' \\
\quad\vdots & \quad\vdots \\
X_m \to \gamma' & X_m^\lambda \to b_m\, \gamma'
\end{array}
$$

Rovnost neterminálů S_k a X_0^λ spolu existencí pravidel uvedených ve druhém sloupci plyne z implikací (3.2), (3.3) a (3.4) na straně 55. Tato pravidla umožňují automatu M' přechod ze stavu $I(«u)$ přes slovo $v/n = b_1 b_2 \ldots b_m$ do nekoncového stavu $I(«uv/n)$ nebo do nějakého koncového stavu. Redukční automat M' je tedy monotónní. □

Příklad 3.2.2. Konstrukci prefixově korektního mon-red-automatu předvedeme pro mon-red-automat M rozpoznávající jazyk jednoduchých aritmetických výrazů. Jeho přechodovou funkci jsme definovali v tabulce 1.1 na straně 14. Tento automat není prefixově korektní, protože po přesunu přes prefix

`a(+a)*+a)`

neskončí v zamítajícím koncovém stavu `ERR`, ale redukuje poslední podslovo `+a`. Prefixově korektní mon-red-automat M' rozpoznávající stejný jazyk jako M získáme tak, že postupně vytvoříme množiny položek uvedené v prvním sloupci tabulky 3.2. Přechodová funkce zkonstruovaného prefixově korektního mon-red-automatu M' je v tabulce 3.3. Tento automat již libovolné slovo začínající nekorektním prefixem `a(+a)*+a)` zamítne v jediné etapě.

3.3 Stavově minimální red-automat

V této podkapitole ukážeme, jak minimalizovat množinu koncových i nekoncových stavů libovolného redukčního automatu při zachování způsobu ukončení jeho etap (akceptace, zamítnutí, redukce) i místa v pracovním seznamu, kde k ukončení etapy dojde. V případě redukce pracovního seznamu bude zachována i redukční posloupnost určující položky seznamu k odstranění.

Dosažitelné stavy. Předpokládejme, že M je libovolný redukční automat a $s \in S_M \cup F_M$ je jeho libovolný stav. Stav s označíme za *dosažitelný*, jestliže se automat M v nějaké etapě přesune do tohoto stavu ze svého počátečního stavu s_M přes nějaké slovo $w \in (\Sigma_M \cup \{»\})^*$, tedy

$$\exists w \in (\Sigma_M \cup \{»\})^* : \delta_M^*(s_M, w) = s.$$

Stav, který není dosažitelný, nazýváme *nedosažitelný*.

Věta 3.3.1. *K libovolnému redukčnímu automatu lze sestrojit ekvivalentní automat, jehož všechny stavy jsou dosažitelné.*

množina položek		následující množina položek
$I(\lambda)$	$S_0 \to \texttt{.}\,\texttt{«}\,S_{10} \mid \texttt{.}\,\texttt{«}\,S_{23}\,S_9$	$I(\texttt{«})$
$I(\texttt{«})$	$S_0 \to \texttt{«}\,\texttt{.}\,S_{10} \mid \texttt{«}\,\texttt{.}\,S_{23}\,S_9,$ $S_{10} \to \texttt{.}\,\texttt{a}\,S_1 \mid \texttt{.}\,\texttt{a}\,S_{14}\,S_{12},$ $S_{23} \to \texttt{.}\,\texttt{(}\,S_{25} \mid \texttt{.}\,\texttt{(}\,S_{23}\,S_{24}$	 $I(\texttt{«a})$ $I(\texttt{«(})$
$I(\texttt{«a})$	$S_{10} \to \texttt{a}\,\texttt{.}\,S_1 \mid \texttt{a}\,\texttt{.}\,S_{14}\,S_{12},$ $S_{14} \to \texttt{.}\,\texttt{+}\,S_{18} \mid \texttt{.}\,\texttt{+}\,S_{23}\,S_{16},$ $S_1 \to \texttt{.}\,\texttt{»}$	 $I(\texttt{«a+})$ $I(\texttt{«a»})$
$I(\texttt{«(})$	$S_{23} \to \texttt{(}\,\texttt{.}\,S_{25} \mid \texttt{(}\,\texttt{.}\,S_{23}\,S_{24},$ $S_{25} \to \texttt{.}\,\texttt{a}\,S_3 \mid \texttt{.}\,\texttt{a}\,S_{13}\,S_{27},$ $S_{23} \to \texttt{.}\,\texttt{(}\,S_{25} \mid \texttt{.}\,\texttt{(}\,S_{23}\,S_{24}$	 $I(\texttt{«(a)})$ $I(\texttt{«(})$
$I(\texttt{«a+})$	$S_{14} \to \texttt{+}\,\texttt{.}\,S_{18} \mid \texttt{+}\,\texttt{.}\,S_{23}\,S_{16},$ $S_{18} \to \texttt{.}\,\texttt{a}\,S_7 \mid \texttt{.}\,\texttt{a}\,S_{14}\,S_{22},$ $S_{23} \to \texttt{.}\,\texttt{(}\,S_{25} \mid \texttt{.}\,\texttt{(}\,S_{23}\,S_{24}$	 $I(\texttt{«a+a})$ $I(\texttt{«(})$
$I(\texttt{«a})$	$S_1 \to \texttt{»}\,\texttt{.}$	
$I(\texttt{«(a)})$	$S_{25} \to \texttt{a}\,\texttt{.}\,S_3 \mid \texttt{a}\,\texttt{.}\,S_{13}\,S_{27},$ $S_{13} \to \texttt{.}\,\texttt{+}\,S_{17} \mid \texttt{.}\,\texttt{+}\,S_{23}\,S_{15},$ $S_3 \to \texttt{.}\,\texttt{)}$	 $I(\texttt{«(a+)})$ $I(\texttt{«(a))})$
$I(\texttt{«a+a})$	$S_{18} \to \texttt{a}\,\texttt{.}\,S_7 \mid \texttt{a}\,\texttt{.}\,S_{14}\,S_{22},$ $S_{14} \to \texttt{.}\,\texttt{+}\,S_{18} \mid \texttt{.}\,\texttt{+}\,S_{23}\,S_{16},$ $S_7 \to \texttt{.}\,\texttt{»}$	 $I(\texttt{«a+})$ $I(\texttt{«a+a»})$
$I(\texttt{«(a+)})$	$S_{13} \to \texttt{+}\,\texttt{.}\,S_{17} \mid \texttt{+}\,\texttt{.}\,S_{23}\,S_{15},$ $S_{17} \to \texttt{.}\,\texttt{a}\,S_5 \mid \texttt{.}\,\texttt{a}\,S_{13}\,S_{21},$ $S_{23} \to \texttt{.}\,\texttt{(}\,S_{25} \mid \texttt{.}\,\texttt{(}\,S_{23}\,S_{24}$	 $I(\texttt{«(a+a)})$ $I(\texttt{«(})$
$I(\texttt{«(a))})$	$S_3 \to \texttt{(}\,\texttt{.}$	
$I(\texttt{«(a+a)})$	$S_{17} \to \texttt{a}\,\texttt{.}\,S_5 \mid \texttt{a}\,\texttt{.}\,S_{13}\,S_{21},$ $S_{13} \to \texttt{.}\,\texttt{+}\,S_{17} \mid \texttt{.}\,\texttt{+}\,S_{23}\,S_{16},$ $S_5 \to \texttt{.}\,\texttt{)}$	 $I(\texttt{«(a+)})$ $I(\texttt{«(a+a))})$
$I(\texttt{«(a+a))})$	$S_5 \to \texttt{)}\,\texttt{.}$	

Tabulka 3.2: Množiny položek.

$f_{M'}$	a	+	(	)	»
$\Rightarrow$ $I(\texttt{«})$	$I(\texttt{«a})$	ERR	$I(\texttt{«(})$	ERR	ERR
$I(\texttt{«a})$	ERR	$I(\texttt{«a+})$	ERR	ERR	ACC
$I(\texttt{«(})$	$I(\texttt{«(a)})$	ERR	$I(\texttt{«(})$	ERR	ERR
$I(\texttt{«a+})$	$I(\texttt{«a+a})$	ERR	$I(\texttt{«(})$	ERR	ERR
$I(\texttt{«(a)})$	ERR	$I(\texttt{«(a+)})$	ERR	RED(101)	ERR
$I(\texttt{«a+a})$	ERR	$I(\texttt{«a+})$	ERR	ERR	RED(110)
$I(\texttt{«(a+)})$	$I(\texttt{«(a+a)})$	ERR	$I(\texttt{«(})$	ERR	ERR
$I(\texttt{«(a+a)})$	ERR	$I(\texttt{«(a+)})$	ERR	RED(110)	ERR

Tabulka 3.3: Přechodová tabulka.

Důkaz. Na redukční automat se můžeme dívat jako na konečný automat s abecedou $\Sigma_M \cup \{»\}$, množinou stavů $S_M \cup F_M \cup \{\texttt{RED}\}$, počátečním stavem s_M, množinou koncových stavů F_M a přechodovou funkcí δ_M. Z teorie konečných automatů tak můžeme převzít konstrukci konečného automatu obsahujícího pouze dosažitelné stavy. □

Relace $\sim$ a etapová ekvivalence. Nechť M a M' jsou libovolné dva redukční automaty se stejnou vstupní abecedou $\Sigma_M = \Sigma_{M'}$. Mezi množinami nekoncových stavů S_M a $S_{M'}$ těchto automatů zavedeme relaci $\sim \subseteq S_M \times S_{M'}$. Říkáme, že nekoncový stav $s \in S_M$ automatu M je v relaci $\sim$ s nekoncovým stavem $s' \in S_{M'}$ automatu M', a píšeme

$$s \sim s',$$

jestliže pro libovolné slovo $w \in (\Sigma_M \cup \{»\})^*$ platí následující implikace:

$$\big(\ \delta^*_M(s,w) \in F_M \quad \text{nebo} \quad \delta^*_{M'}(s',w) \in F_M\ \big) \implies \delta^*_M(s,w) = \delta^*_M(s',w)$$

Je-li $M = M'$, pak relace $\sim$ je ekvivalencí na množině nekoncových stavů S_M automatu M. Nekoncové stavy, jež jsou spolu v relaci $\sim$ nazveme *etapově ekvivalentními*. Stejně nazveme i redukční automaty, jejichž počáteční stavy jsou v relaci $\sim$. Pro dva etapově ekvivalentní redukční automaty platí, že libovolné slovo buď oba přijmou, nebo oba zamítnou, nebo oba redukují, a to ve všech třech případech na stejném místě pracovního seznamu a v případě redukce i podle stejné redukční posloupnosti. Libovolné dva etapově ekvivalentní redukční automaty jsou tedy zřejmě i silně ekvivalentní[3] a naopak. Protože etapová ekvivalence i silná ekvivalence pojmenovávají stejný jev, budeme dále používat jen pojem silná ekvivalence.

Stavová minimalita. Nechť M je libovolný redukční automat. Automat M je *stavově minimální*, jestliže

- všechny jeho stavy jsou dosažitelné a
- žádné dva jeho různé nekoncové stavy nejsou ekvivalentní.

Redukční automat M' nazveme *reduktem* redukčního automatu M, jestliže

- M' je silně ekvivalentní s M a
- M' je stavově minimální.

Věta 3.3.2. *K libovolnému redukčnímu automatu lze sestrojit jeho redukt.*

Důkaz. Nechť M je redukční automat. Ukážeme, jak k automatu M sestrojit jeho redukt M'.

Úlohu sestrojit redukt redukčního automatu převedeme na úlohu sestrojit redukt Mooreova stroje, která je již vyřešená v literatuře zabývající se teorií konečných automatů a konečně stavových převodníků.

Na redukční automat se můžeme dívat jako na konečný automat rozšířený o možnost redukovat pracovní seznam. Redukční automat M počítá nad slovem

[3] Silnou ekvivalenci jsme definovali na straně 12.

obsaženým v pracovním seznamu přechodovou funkci pomocí které určí, zda a kde slovo přijmout, zamítnout nebo redukovat, a v případě redukce i jak ho redukovat. Mooreův stroj je v zásadě také konečným automatem rozšířeným o značkování vstupního slova. Nad vstupním slovem rovněž počítá přechodovou funkci a na základě její hodnoty průběžně značkuje vstupní slovo. Redukt A' Mooreova stroje A je „stavově minimální“ stroj, který libovolné slovo označkuje stejně jako původní stroj A. Jde tedy o to pro redukční automat M navrhnout Mooreův stroj A, který svou značkovací funkcí ve vstupním slově určí zda a kde ho přijmout, zamítnout nebo redukovat (a jak ho redukovat) stejně jako redukční automat M. Potom ke stroji A sestrojíme jeho redukt A' (se stejným chováním jako A) a ten nakonec převedeme zpět na redukční automat M'. Náš postup bude tedy následující:

1. Redukční automat M převedeme na Mooreův stroj A.

2. Sestrojíme redukt A' Mooreova stroje A.

3. Redukt A' převedeme zpět na redukční automat M'.

Mooreův stroj A sestrojíme následujícím způsobem: Množinou S_A jeho stavů je množina všech koncových a nekoncových stavů automatu M a pomocného stavu `RED`. Počáteční stav s_A se shoduje s počátečním stavem s_M redukčního automatu. Vstupní abeceda Σ_A je tvořená abecedou Σ_M automatu M a pravým omezovačem ». Výstupní abeceda Γ_A se skládá z počátečního stavu s_M, všech koncových stavů automatu M a pomocného symbolu `RED`. Jeho přechodová funkce δ_A je totožná se zobecněnou přechodovou funkcí δ_M redukčního automatu M. Jeho značkovací funkce μ_A libovolný nekoncový stav automatu M označkuje počátečním stavem automatu M a libovolný koncový nebo pomocný stav automatu M označkuje sebou samým.

$$
\begin{aligned}
S_A &= S_M \cup F_M \cup \{\texttt{RED}\} \\
s_A &= s_M \\
\Sigma_A &= \Sigma_M \cup \{»\} \\
\Gamma_A &= F_M \cup \{\texttt{RED}\} \cup \{s_M\} \\
\delta_A &= \delta_M \\
\mu_A(s) &= \begin{cases} s_M, & \text{je-li } s \in S_M \\ s, & \text{je-li } s \in F_M \cup \{\texttt{RED}\} \end{cases}
\end{aligned}
$$

K takto definovanému Mooreovu stroji A sestrojíme jeho redukt A' s přechodovou funkcí $\delta_{A'}$ a značkovací funkcí $\mu_{A'}$ konstrukcí uvedenou například v [15].

Nakonec od reduktu A' přejdeme zpět k redukčnímu automatu M'. Jeho abeceda se shoduje s abecedou původního redukčního automatu M. Množina jeho stavů $S_{M'}$ je tvořena právě všemi stavy stroje A' označkovanými počatečním stavem s_M původního redukčního automatu M. Množinou jeho koncových stavů je obor hodnot značkovací funkce $\mu_{A'}$ reduktu A' bez s_M a bez pomocného stavu `RED`. Hodnota přechodové funkce $f_{M'}$ výsledného automatu M' pro stav $s \in S_{M'}$ a symbol $a \in \Sigma_{M'} \cup \{»\}$ je rovna stavu $s' = \delta_{A'}(s, a)$, pokud $\mu_{A'}(s') = s_M$ původního

automatu M, jinak je rovna $\mu_{A'}(s')$.

$$\begin{aligned}
\Sigma_{M'} &= \Sigma_{A'} \setminus \{»\} = \Sigma_A \setminus \{»\} = \Sigma_M \\
S_{M'} &= \{s \mid \mu_{A'}(s) = s_M\} \\
s_{M'} &= s_{A'} = s_A = s_M \\
F_{M'} &= \{s' \in F_M \mid \exists s \in S_{A'} : s' = \mu_{A'}(s)\} \\
f_{M'}(s,a) &= \begin{cases} s', & \text{je-li } s' = \delta_{A'}(s,a) \in S_{M'} \\ \mu_{A'}(s'), & \text{je-li } s' = \delta_{A'}(s,a) \notin S_{M'} \end{cases}
\end{aligned}$$

Nyní není těžké nahlédnout, že zkonstruovaný redukční automat M' je reduktem automatu M.

Silná ekvivalence redukčních automatů M a M' plyne z následujících faktů:

1. Množina $F_{M'}$ koncových stavů redukčního automatu M' obsahuje právě všechny dosažitelné koncové stavy redukčního automatu M.

2. Počáteční stav $s_{A'}$ Mooreova stroje A' je behaviorálně ekvivalentní s počátečním stavem s_A Mooreova stroje A.

Pro libovolné slovo $w \in \Sigma_M^* \cup \Sigma_M^*»$ a dosažitelný redukční koncový stav $\texttt{RED}(n)$ redukčního automatu M tedy platí následující ekvivalence:

$$\begin{aligned}
\delta^*_{M'}(s_{M'}, w) = \texttt{RED}(n) &\iff \\
\mu_{A'}\big(\delta^*_{A'}(s_{A'}, w)\big) = \texttt{RED}(n) &\iff \\
\mu_A\big(\delta^*_A(s_A, w)\big) = \texttt{RED}(n) &\iff \\
\delta^*_M(s_M, w) = \texttt{RED}(n) &
\end{aligned}$$

Stejné ekvivalence zřejmě platí, když redukční koncový stav $\texttt{RED}(n)$ nahradíme příjímajícím resp. zamítajícím koncovým stavem `ACC` resp. `ERR`.

Stavová minimalita automatu M' přímo plyne z redukovanosti Mooreova stroje A' a z postupu, kterým jsme ze stroje A' získali automat M'.

Redukční automat M' je tedy reduktem redukčního automatu M. □

Abychom mohli porovnávat množiny stavů silně ekvivalentních redukčních automatů, zavedeme mezi nimi isomorfismus – vzájemně jednoznačné zobrazení množin jejich stavů, které respektuje jejich přechodové funkce i způsoby ukončení etap. Přijímající koncový stav `ACC` zobrazuje na `ACC`, zamítající koncový stav `ERR` zobrazuje na `ERR` a každý redukční koncový stav $\texttt{RED}(n)$ zobrazuje na redukční stav se stejnou redukční posloupností. Nad množinou koncových stavů je tedy identitou.

Říkáme, že redukční automaty M a M' se stejnou abecedou $\Sigma_M = \Sigma_{M'}$ jsou *isomorfní*, jestliže existuje vzájemně jednoznačné zobrazení h mezi množinami $S_M \cup F_M$ a $S_{M'} \cup F_{M'}$ jejich koncových i nekoncových stavů,

$$h : S_M \cup F_M \longrightarrow S_{M'} \cup F_{M'},$$

pro které zároveň platí následující podmínky:

- pro libovolný koncový stav $s \in F_M$ je $h(s) = s$,

- pro libovolný nekoncový stav $s \in S_M$, symbol $a \in \Sigma_M \cup \{»\}$ a koncový nebo nekoncový stav $s' \in S_M \cup F_M$ je

$$\delta_M(s, a) = s', \quad \text{právě když} \quad \delta_{M'}\big(h(s), a\big) = h(s').$$

Věta 3.3.3. *Libovolné dva redukty téhož redukčního automatu jsou isomorfní.*

Tuto větu lze zřejmě dokázat modifikací důkazu stejné věty pro Mooreovy stroje.

Důsledek 3.3.4. *Žádný redukční automat nemá méně stavů než jeho redukt.*

3.4 Shrnutí

V této kapitole jsme ukázali, jak k libovolnému monotónnímu redukčnímu automatu sestrojit ekvivalentní LR(0)-gramatiku. Tu jsme využili při konstrukci ekvivalentního prefixově korektního mon-red-automatu. Dále jsme ukázali, jak k libovolnému redukčnímu automatu (monotonnie není v tomto případě nutná) sestrojit silně ekvivalentní redukční automat s minimální množinou koncových i nekoncových stavů. Silná ekvivalence zaručuje, že konstrukce stavově minimálního redukčního automatu zachovává jak prefixovou korektnost, tak i monotonii. Pokud tedy k mon-red-automatu M nejprve zkonstruujeme ekvivalentní prefixově korektní mon-red-automat M' a k němu pak silně ekvivalentní stavově minimální redukční automat M'', máme jistotu, že M'' zůstane prefixově korektní i monotónní.

Závěr

Jedním ze záměrů předložené práce bylo teoreticky podložit známou, často prakticky používanou metodu zotavení, tzv. metodu panického módu. Až doposud jsme k tomuto účelu používali pojem hlavičkových symbolů (či hlaviček), a panický mód byl modelován různými typy automatů. Z prací zabývajícími se hlavičkami je této práci nejbližší publikace [26], kde je uveden svou podstatou velmi speciální případ metody představené ve druhé kapitole této práce, anglicky popsané též v [28]. Detailnější souvislosti mezi těmito dvěma metodami a dalšími metodami založenými na jiném pojetí syntaktických chyb než v [28] plánujeme popsat v blízké budoucnosti. Metodu popsanou v [28], vzhledem k její univerzálnosti pro celou třídu deterministických bezkontextových jazyků, považujeme za vhodné východisko pro další výzkum v této oblasti.

Jedno z možných nových pojetí syntaktických chyb je naznačeno ve druhé příloze k této práci, kterou citujeme jako [27]. Pracuje se tam s pojmem „pumpovacích nekonzistencí“. Takové nekonzistence lze jednoduše a jednoznačně opravovat a jejich opravou lze například spojovat dvě sousedící nekorektní jádra v jediný korektní úsek.

Definice pumpovacích (ne)konzistencí lze přirozeně zesílit tak, že pro metody lokalizce chyb založené na tomto pojmu nebude nutné předpokládat monotonii a prefixovou korektnost. Tímto směrem by se měl ubírat v nebližší době náš další výzkum.

Seznam použité literatury

[1] Aho, Alfred V., and Ullman, Jeffrey D. *The Theory of Parsing, Translation, and Compiling.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, Volume I, 1972.

[2] Aho, Alfred V., and Ullman, Jeffrey D. *The Theory of Parsing, Translation, and Compiling.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, Volume II, 1973.

[3] Cormack, Gordon V. *An LR substring parser for noncorrecting syntax error recovery.* ACM SIGPLAN Notices, vol. 24, no. 7, pp. 161–169, July 1989.

[4] Chytil, Michal. *Teorie automatů a formálních jazyků.* SPN Praha, 1978.

[5] Chytil, Michal. *Automaty a gramatiky.* SNTL Praha, 1984.

[6] Deudekom, Arthur van, and Kooiman, Peter. *Top-down non-correcting error recovery in LLgen.* Report IR-338, Vrije Universiteit Amsterdam, 1993.

[7] Deudekom, Arthur van, Grune, Dick, and Kooiman, Peter. *Initial experience with noncorrecting syntax error handling.* Informatica rapport, Vrije Universiteit, Faculteit der Wiskunde en Informatica, 1993.

[8] Jančar, Petr. *Nondeterministic Forgetting Automata are Less Powerful than Deterministic Linear Bounded Automata.* Acta Math. et Inf. Univ. Ostraviensis, 1, Ostrava, pp. 67–73, 1993.

[9] Jančar, Petr, Mráz, František, and Plátek Martin. *A Taxonomy of Forgetting Automata.* In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, Mathematical Foundations of Computer Science 1993, 18th International Symposium, volume 711 of LNCS, pp. 527–536, Springer, Gdansk, Poland, 30 August - 3 September 1993.

[10] Jančar, Petr, Mráz, František, Plátek, Martin, Procházka, Martin, and Vogel, Jörg. *Restarting Automata, Marcus Grammars and Context-Free Languages.* In J. Dassow, G. Rozenberg, and A. Salomaa, editors, Developments in Language Theory II, World Scientific Publ., pp. 102–111, Magdeburg, Germany, 1996.

[11] Jančar, Petr, Mráz, František, Plátek, Martin, and Vogel, Jörg. *Restarting Automata.* In Horst Reichel, editor, FCT'95, volume 965 of LNCS, pp. 283–292, Dresden, Germany, Springer, August 1995.

[12] Jančar, Petr, Mráz, František, Plátek, Martin, Procházka, Martin, and Vogel, Jörg. *Deleting Automata with a Restart Operation.* In S. Bozapalidis, editor, Developments in Language Theory, 1997, pp. 191–202, Thessaloniki, Greece, University of Thessaloniki, 1998.

[13] Kuboň, Vladislav, Plátek, Martin. *A Grammar Based Approach to a Grammar Checking of Free Word Order Languages.* In Proceedings of the 15th conference on Computational linguistics, Vol. 2, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 906–910, 1994.

[14] Melichar, Bořivoj. *Překladače.* Ediční středisko Českého vysokého učení technického, 1983.

[15] Moore, Edward F. *Gedanken-experiments on Sequential Machines.* Automata Studies, Annals of Mathematical Studies, 34, pp. 129-–153. Princeton University Press, Princeton, N.J., 1956.

[16] Mráz, František, Plátek, Martin, and Procházka, Martin. *Deleting automata with a restart operation and Marcus grammars.* In S. Bozapalidis, editor, Developments in Language Theory, 1997, pp. 329–342, University of Thessaloniki, Thessaloniki, Greece, 1998.

[17] Mráz, František, Plátek, Martin, and Procházka, Martin. *On special forms of restarting automata.* In V. Mitrana, C. Martin-Vide, editors, Words, sequences, grammars, languages: where biology, computer science, linguistics and mathematics meet I, pp. 149–160, Kluwer, Dordrecht, 2000.

[18] Mráz, František, Plátek, Martin, and Procházka, Martin. *Restarting automata, deleting, and Marcus grammars.* In C. Martin-Vide and Gh. Paun, editors, Recent topics in mathematical and computational linguistics, pp. 218–233, The Publishing House of the Romanian Academy, Bucharest, 2000.

[19] Panevová, Jarmila, Straňáková, Markéta. *Some types of syntactic ambiguity; how to treat them in an automatic procedure.* In Workshop on Text, Speech and Dialog (TSD'99), Plzeň, Czech Republic, September 1999.

[20] Plátek, Martin. *Syntactic Error Recovery with Formal Guarantees I.* Technical Report No. 100, Dept. of Theoretical Computer Science, Faculty of Mathematics and Physics, Charles University, Prague, 1992.

[21] Plátek, Martin. *Construction of a Robust Parser from a Deterministic Reduced Parser.* In Kybernetika, Vol. 33, Number 3, pp. 311–332, 1997.

[22] Plátek, Martin. *Weak Cyclic Forms of Restarting Automata.* In Rozenberg, G. and Thomas, W., editors, Developments in Languages Theory - Foundations, Applications, and Perspectives, Proceedings DLT 1999, World Scientific, Singapore, pp. 115–124, 2000.

[23] Procházka, Martin. *Syntaktické chyby a jejich vyhledávání.* diplomová práce, Univerzita Karlova v Praze, fakulta Matematicko-fyzikální, Praha, 1994.

[24] Procházka, Martin. *Concepts of Syntax Error Recovery for Monotonic Reducing Automata.* In MIS'2004, matfyzpress, Praha, 2004.

[25] Procházka, Martin, Plátek, Martin. *Redukční automaty, monotonie a redukovanost.* In Proceedings of ITAT 2002, Informations Technologies - Applications and Theory, Malino - Brdo, Slovakia, Invited Lecture, pp. 23—32, 2003.

[26] Procházka, Martin, Plátek, Martin. *Redukční automaty a syntaktické chyby.* Sborník konference ITAT 2011 (in Czech), Lopatková Markéta (editor), Information Technologies – Applications and Theory, Univerzita Pavla Jozefa Šafárika v Košiciach, ISBN 978-80-89557-01-1, 2011, pp. 23-30.

[27] Procházka, Martin, Plátek, Martin. *A Taxonomy of Reducing Automata Motivated by Error Localization.* Submitted to the Workshop on Non-Classical Models of Automata and Applications, University of Fribourg, 2012.

[28] Procházka, Martin, Plátek, Martin. *Localization of (In)Consistencies by Monotone Reducing Automata.* Submitted to the conference ITAT 2012.

[29] Richter, Helmut. *Noncorrecting Syntax Error Recovery.* ACM Trans. Prog. Lang. Sys., vol.7, no.3, pp. 478–489, July 1985.

Seznam tabulek

1.1 Přechodová tabulka. 14
1.2 Stavy jako regulární výrazy. 15

2.1 Přechodová tabulka automatu M_1. 26
2.2 Přechodová tabulka automatu M_2. 26
2.3 Přechodová tabulka automatu M_3. 26
2.4 Funkce postprefixového robustního analyzátoru. 32
2.5 Redukovatelná podslova automatu M_1. 41
2.6 Redukovatelná podslova automatu M_2. 41

3.1 Grafická reprezentace pravidel gramatiky. 50
3.2 Množiny položek. 67
3.3 Přechodová tabulka. 67

Seznam obrázků

1.1 Redukční automat. 8
1.2 Zkrácení pracovního seznamu. 9
1.3 Přechodový graf. 15
1.4 Redukční a akceptační pravidla. 15

2.1 (Ne)korektní jádra a (post)prefixové korektnosti. 20
2.2 Analýza slova «`a++(a)+)(a`» robustním analyzátorem A_1. 27
2.3 Analýza slova «`a++(a)+)(a`» robustním analyzátorem A_2. 28
2.4 Analýza slova «`a++(a)+)(a`» robustním analyzátorem A_3. 29

3.1 Neterminály gramatiky z příkladu 3.1.1. 52
3.2 Pravidla gramatiky z příkladu 3.1.1. 53
3.3 Derivační strom, který dává slovo «`a+a+((a))`». 53
3.4 Funkce R. 55

Přílohy

1. Procházka, Martin. *Concepts of Syntax Error Recovery for Monotonic Reducing Automata*. In MIS'2004, matfyzpress, Praha, 2004.

2. Procházka, Martin, Plátek, Martin. *A Taxonomy of Reducing Automata Motivated by Error Localization*. Submitted to the Workshop on Non-Classical Models of Automata and Applications, University of Fribourg, 2012.