Týden 6

Přednáška

‘Překladač’ sestrojující k regulárnímu výrazu ekvivalentní konečný automat

Nejdříve jsme si z minula připomněli jednoznačnou gramatiku G pro jazyk $RV(\{a, b\})$

$$
\begin{aligned}
R &\longrightarrow T + R \mid T \\
T &\longrightarrow FT \mid F \\
F &\longrightarrow F^* \mid (R) \mid C \\
C &\longrightarrow a \mid b
\end{aligned}
$$

v níž jsme ale pro zjednodušení vynechali symboly $\emptyset$ a ε. Naše G tedy generuje jazyk v abecedě $\Sigma = \{a, b, +, ^*, (,)\}$.
Jako příklad řetězce z $L(G)$ jsme opět vzali

$$(aa + b)^* .$$

Poté jsme provedli dále popsanou konstrukci derivačního stromu pro slovo $(aa + b)^*$; jedná se o konstrukci typu zdola-nahoru (strom konstruujeme postupně od listů ke kořeni).
Účelem samozřejmě nebylo jen zkonstruovat tento konkrétní strom (který díky jednoduchosti vstupního řetězce jistě zkonstruujete téměř bez přemýšlení), ale alespoň naznačit algoritmickou metodu, která stojí v pozadí reálné syntaktické analýzy v překladačích. Po zkonstruování stromu si ukážeme, jak lze takovouto syntaktickou strukturu využít pro generování „cílového kódu“, čímž v našem případě rozumíme konečný automat přijímající jazyk, který je reprezentován vstupním regulárním výrazem.

Konstrukce derivačního stromu zdola-nahoru

Na vstupním řetězci se pohybuje čtecí hlava (jen doprava). Na začátku stojí na nejlevějším symbolu, její pozici znázorněme podtržením:

$$\underline{(}aa + b)^*$$

Budeme využívat datovou strukturu zásobník; jednotlivá položka ukládaná na zásobník (odpovídá vrcholu derivačního stromu a) obsahuje jeden terminál či neterminál gramatiky G. Položky s neterminály budou navíc obsahovat ukazatele (pointers) do paměti (typu halda, v níž dynamicky alokujeme potřebné ‘buňky’). Na začátku jsou zásobník i paměť prázdné.
Jedna z ‘instrukcí’, kterou aplikujeme, pokud jsou splněny její předpoklady, zní takto:

> **Instrukce 1**
>
> Jestliže zásobník neobsahuje na vrcholu pravou stranu nějakého pravidla gramatiky G a nebylo-li přečteno celé vstupní slovo, přesune se do zásobníku aktuálně čtený symbol ze vstupu a čtecí hlava se posune (o jedno pole doprava).

Na začátku tedy je Instrukce 1 aplikovatelná a po jejím provedení dostaneme následující konfiguraci; zde druhý řádek ukazuje obsah zásobníku (vlevo je vždy dno zásobníku, vpravo jeho vrchol). Jednu položku na zásobníku vždy ohraničujeme závorkami [ ]; ty nejsou symboly naší gramatiky G, takže si je s nimi nespleteme.

$(\underline{a}a + b)^*$
$[(]$

Znovu je aplikovatelná Instrukce 1, dojde tedy k dalšímu přesunu a následující konfigurace je

$(a\underline{a} + b)^*$
$[(][a]$

Nyní vrchní symboly zásobníku, v našem případě jeden, tvoří pravou stranu pravidla gramatiky, konkrétně pravidla $C \longrightarrow a$. Jedná se o terminální symbol, který se pochopitelně musí vyskytovat jako list v konstruovaném derivačním stromě. Je snadné nahlédnout, že předchůdce tohoto listu musí být v každém případě označen C a musí mít onen list označený a jako jediného následníka. Proto nemůžeme nic pokazit provedením tzv. redukce podle pravidla $C \longrightarrow a$. Obecně vypadá (procedura) redukce následovně.

Redukce podle pravidla $X \longrightarrow Y_1Y_2 \ldots Y_n$
(Bude se používat jen v případě, že zásobník má na vrcholu položky $pol_1, pol_2, \ldots, pol_n$ (kde pol_n je ta nejvrchnější) obsahující postupně symboly $Y_1, Y_2, \ldots, Y_n$; zde Y_i mohou být neterminály i terminály. K redukci ale nedojde v takovém případě automaticky, budou muset být případně splněny další podmínky.)

Nejprve se alokuje nová paměť: n buněk, do nichž jsou uloženy (kompletní) položky $pol_1, pol_2, \ldots, pol_n$; tyto položky se ze zásobníku odstraní a na vrchol zásobníku se vloží nová položka se symbolem X a n ukazateli (pointery) $p_1, p_2, \ldots, p_n$, které ukazují na nově alokované buňky: ukazatel p_i ukazuje na buňku (tedy je adresou buňky), do které byla uložena položka pol_i.

V našem případě tedy zformulujeme tuto instrukci:

Instrukce 2

Jestliže je na vrcholu zásobníku a, provedeme redukci podle pravidla $C \longrightarrow a$.

Její provedení v aktuální konfiguraci vyústí v novou konfiguraci

$(a\underline{a} + b)^*$
$[(][C, p_1]$
$p_1 : [a]$

Kromě vstupu a zásobníku teď už konfigurace zahrnuje i kousek haldy. Tam žádnou (viditelnou) strukturu nepředpokládáme, byť se pro názornost budeme snažit obsah zapisovat tak, ať je v zápisu vznikající derivační strom trochu 'vidět'. Můžete si samozřejmě dokreslovat příslušné šipky; zde by šlo o šipku, která znázorní, že položka $[C, p_1]$ ukazuje na paměťovou buňku (s adresou) p_1.

Vidíme teď, že opět máme pravou stranu nějakého pravidla na vrcholu zásobníku, konkrétně jde o pravidlo $F \longrightarrow C$. Jelikož C se jinde na pravé straně nevyskytuje, můžeme bezpečně používat následující instrukci:

Instrukce 3

Jestliže je na vrcholu zásobníku C, provedeme redukci podle pravidla $F \longrightarrow C$.

Její aplikací dostaneme konfiguraci

$(a\underline{a} + b)^*$
$[(][F, p_2]$
$p_2 : [C, p_1]$
$p_1 : [a]$

(Alokovali jsme samozřejmě novou buňku paměti, což je znázorněno hodnotou p_2, která dosud nebyla použita.)

Teď musíme být opatrní. Na vrcholu zásobníku je sice pravá strana nějakého pravidla, konkrétně $T \longrightarrow F$, ale F se vyskytuje i na pravých stranách jiných pravidel, konkrétně $T \longrightarrow FT$ a $F \longrightarrow F^*$. Není těžké vytušit, že by měly fungovat následující instrukce. ('Fungovat' znamená, že aplikace instrukce nemůže způsobit to, že konstrukce derivačního stromu zhavaruje, ač vstupní slovo je v $L(G)$. My se zde v naší ilustraci omezujeme na ono 'vytušení', ve skutečnosti bychom příslušné tvrzení samozřejmě museli dokázat.)

Instrukce 4

Jestliže je na vrcholu zásobníku F a aktuální čtený symbol je *, provedeme přesun čteného symbolu (* jde do zásobníku a čtecí hlava se posune) a pak redukci podle pravidla $F \longrightarrow F^*$.

Instrukce 5

Jestliže je na vrcholu zásobníku F a aktuální čtený symbol je + nebo), nebo je vstupní slovo již přečteno (čtecí hlava stojí za ním), pak provedeme redukci podle pravidla $T \longrightarrow F$.

Instrukce 6

Jestliže je na vrcholu zásobníku F a aktuální čtený symbol je a, b, nebo (, pak provedeme přesun (vstupního symbolu do zásobníku). (Položka s F bude 'čekat', až se v budoucnu objeví napravo od ní T jako vrchol zásobníku.)

V našem příkladu je tedy aplikována Instrukce 6, což vede ke konfiguraci

$$
\begin{array}{l}
(aa\underline{+}b)^* \\
[(][F, p_2][a] \\
p_2 : [C, p_1] \\
p_1 : [a]
\end{array}
$$

Na vrcholu se objevilo a, takže aplikacemi Instrukcí 2 a 3 se dostaneme do konfigurace

$$
\begin{array}{l}
(aa\underline{+}b)^* \\
[(][F, p_2][F, p_4] \\
p_2 : [C, p_1] \quad p_4 : [C, p_3] \\
p_1 : [a] \quad p_3 : [a]
\end{array}
$$

Nyní se uplatní Instrukce 5 a dostáváme

$$
\begin{array}{l}
(aa\underline{+}b)^* \\
[(][F, p_2][T, p_5] \\
\qquad\qquad p_5 : [F, p_4] \\
p_2 : [C, p_1] \quad p_4 : [C, p_3] \\
p_1 : [a] \quad p_3 : [a]
\end{array}
$$

Máme tedy na vrcholu zásobníku pravou stranu pravidla $T \longrightarrow FT$; opět lze vytušit (a lze pečlivě dokázat), že můžeme bezpečně používat následující instrukci:

Instrukce 7

Jestliže je na vrcholu zásobníku FT, zredukujeme podle pravidla $T \longrightarrow FT$.

Při její aplikaci teď poprvé redukujeme podle pravidla, u nějž je délka pravé strany větší než 1. Výsledkem bude konfigurace

$$
\begin{array}{l}
(aa\underline{+}b)^* \\
[(][T, p_6, p_7] \\
p_6 : [F, p_2] \quad p_7 : [T, p_5] \\
\qquad\qquad p_5 : [F, p_4] \\
p_2 : [C, p_1] \quad p_4 : [C, p_3] \\
p_1 : [a] \quad p_3 : [a]
\end{array}
$$

Nyní je na vrcholu zásobníku T, ale není tam FT. Pro budoucí redukci zahrnující ono T, které je momentálně na vrcholu, připadají tedy v úvahu pravidla $R \longrightarrow T + R$, $R \longrightarrow T$. Zase se (dá ukázat, že se) tento konflikt dá vyřešit pomocí aktuálně čteného symbolu:

Instrukce 8

Jestliže je na vrcholu zásobníku T (ale ne FT) a aktuální čtený symbol je $+$, provedeme přesun čteného symbolu ($+$ jde do zásobníku a čtecí hlava se posune).

Instrukce 9

Jestliže je na vrcholu zásobníku T (ale ne FT) a aktuální čtený symbol není $+$, provedeme redukci podle pravidla $R \longrightarrow T$.

Poznámka. Jeden příklad na cvičení žádá, ať zjistíte, jaké situace mohou nastat, když je na vstupu správně utvořený regulární výraz (tedy slovo z $L(G)$), na vrcholu zásobníku je T a aktuální čtený symbol není $+$. Toho lze využít tak, že při zjištění 'nelegální' situace může algoritmus ihned skončit zahlášením chyby (což znamená, že slovo na vstupu nepatří do $L(G)$).

V našem konkrétním případě je tedy aplikována Instrukce 8. Po ní je aplikovatelná Instrukce 1, takže dojde k dalšímu přesunu. Na vrcholu se ocitne b, pro které pochopitelně použijeme analogickou instrukci k Instrukci 2, což je

Instrukce 10

Jestliže je na vrcholu zásobníku b, provedeme redukci podle pravidla $C \longrightarrow b$.

Pak se uplatní instrukce 3, takže po této sérii skončíme v konfiguraci

$$
\begin{array}{l}
(aa + b)^* \\
[(][T, p_6, p_7][+][F, p_9] \\
p_6 : [F, p_2] \quad p_7 : [T, p_5] \\
\qquad\qquad p_5 : [F, p_4] \\
p_2 : [C, p_1] \quad p_4 : [C, p_3] \quad p_9 : [C, p_8] \\
p_1 : [a] \quad p_3 : [a] \qquad\qquad p_8 : [b]
\end{array}
$$

Jelikož další čtený symbol je pravá závorka, uplatní se Instrukce 5 a následně Instrukce 9. Dostaneme tedy

$$
\begin{array}{l}
(aa + b)^* \\
[(][T, p_6, p_7][+][R, p_{11}] \\
p_6 : [F, p_2] \quad p_7 : [T, p_5] \quad p_{11} : [T, p_{10}] \\
\qquad\qquad p_5 : [F, p_4] \quad p_{10} : [F, p_9] \\
p_2 : [C, p_1] \quad p_4 : [C, p_3] \quad p_9 : [C, p_8] \\
p_1 : [a] \quad p_3 : [a] \qquad\qquad p_8 : [b]
\end{array}
$$

Máme na vrcholu pravou stranu pravidla $R \longrightarrow T + R$; opět by šlo ověřit bezpečnost následující instrukce:

Instrukce 11

Jestliže je na vrcholu zásobníku $T+R$, provedeme redukci podle pravidla $R \longrightarrow T + R$.

Po její aplikaci dostáváme

$(aa + b)^*$
$[(][[R, p_{12}, p_{13}, p_{14}]$

$p_{12} : [T, p_6, p_7] \quad p_{14} : [R, p_{11}]$
$p_6 : [F, p_2] \quad p_7 : [T, p_5] \quad p_{11} : [T, p_{10}]$
$p_5 : [F, p_4] \quad p_{10} : [F, p_9]$
$p_2 : [C, p_1] \quad p_4 : [C, p_3] \quad p_9 : [C, p_8]$
$p_1 : [a] \quad p_3 : [a] \quad p_{13} : [+] \quad p_8 : [b]$

(Samozřejmě je jedno, kam např. buňku s + v znázornění haldy zakreslíme. Pro přehlednost obrázku je samozřejmě užitečné ji nakreslit na příslušné místo do "dolní (terminální) řady".)
Nyní se opět uplatní Instrukce 1 a potom následující zřejmá instrukce:

Instrukce 12

Jestliže je na vrcholu zásobníku (R), provedeme redukci podle pravidla $F \longrightarrow (R)$.

Výsledek bude

$(aa + b)^*$
$[F, p_{15}, p_{16}, p_{17}]$

$p_{16} : [R, p_{12}, p_{13}, p_{14}]$
$p_{12} : [T, p_6, p_7] \quad p_{14} : [R, p_{11}]$
$p_6 : [F, p_2] \quad p_7 : [T, p_5] \quad p_{11} : [T, p_{10}]$
$p_5 : [F, p_4] \quad p_{10} : [F, p_9]$
$p_2 : [C, p_1] \quad p_4 : [C, p_3] \quad p_9 : [C, p_8]$
$p_{15} : [(] \quad p_1 : [a] \quad p_3 : [a] \quad p_{13} : [+] \quad p_8 : [b] \quad p_{17} : [)]$

Nyní se uplatní Instrukce 4 a po ní Instrukce 5 a pak Instrukce 9. Výsledek bude konfigurace

$(aa + b)^*$
$[R, p_{21}]$

$p_{21} : [T, p_{20}]$
$p_{20} : [F, p_{18}, p_{19}]$
$p_{18} : [F, p_{15}, p_{16}, p_{17}]$

$p_{16} : [R, p_{12}, p_{13}, p_{14}]$
$p_{12} : [T, p_6, p_7] \quad p_{14} : [R, p_{11}]$
$p_6 : [F, p_2] \quad p_7 : [T, p_5] \quad p_{11} : [T, p_{10}]$
$p_5 : [F, p_4] \quad p_{10} : [F, p_9]$
$p_2 : [C, p_1] \quad p_4 : [C, p_3] \quad p_9 : [C, p_8]$
$p_{15} : [(] \quad p_1 : [a] \quad p_3 : [a] \quad p_{13} : [+] \quad p_8 : [b] \quad p_{17} : [)] \quad p_{19} : [^*]$

v níž je celé vstupní slovo přečtené a v zásobníku je jen položka s počátečním neterminálem R. Proběhl takto úspěšný výpočet, který sestrojil derivační strom pro slovo $(aa+b)^*$. Kořen stromu je ona položka $[R, p_{21}]$ v zásobníku.

Mělo by být všem zřejmé, že pokud výpočet pro nějaké vstupní slovo w takto úspěšně skončí (celé slovo přečteno a v zásobníku je jen položka obsahující počáteční neterminál), tak opravdu sestrojil derivační strom pro slovo w, podle gramatiky G, a tedy nutně $w \in L(G)$. Opačný směr, že totiž naše instrukce jsou bezpečné a nemohou zapříčinit neúspěšný výpočet pro nějaké $w \in L(G)$, už tak zřejmý není a museli bychom jej pečlivě dokázat, jak jsme zmiňovali v průběhu. I na takové ověření, a celou konstrukci našich instrukcí, ovšem existují algoritmy, ale tak daleko zde už nepůjdeme.

Sestrojení konečného automatu na základě derivačního stromu

V této části jen stručně dotáhneme, co jsme avízovali. Konstrukce derivačního stromu nebyla samoúčelná a neslouží tak jen ke zjištění, zda zadané slovo je generováno příslušnou gramatikou. Kořen zkonstruovaného stromu (položku s R v zásobníku po skončení úspěšného výpočtu) totiž můžeme předložit funkci NFA, která sestrojí odpovídající konečný automat – stačí nám nedeterministický s případnými ε-šipkami. Níže uvedená definice funkce snad nevyžaduje bližší komentář.

automat NFA(node v)
(* procedura vracející k zadanému vrcholu v derivačního stromu automat, např. ve formě tabulky, který odpovídá podvýrazu určenému podstromem s kořenem v *)
{

if ($v.symb =$ 'a') return

	a	b	ε
$\rightarrow q_1$	q_2	-	-
(q_2)	-	-	-

;

if ($v.symb =$ 'b') return

	a	b	ε
$\rightarrow q_1$	-	q_2	-
(q_2)	-	-	-

;

if ($v.symb =$ 'R' and $v.rule =$ "$R \longrightarrow R + T$")
return UNION(NFA($v.succ_1$), NFA($v.succ_3$));
if ($v.symb =$ 'T' and $v.rule =$ "$T \longrightarrow FT$") return CONC(NFA($v.succ_1$), NFA($v.succ_2$));
if ($v.symb =$ 'F' and $v.rule =$ "$F \longrightarrow F^*$") return ITER(NFA($v.succ_1$));
if ($v.symb =$ 'F' and $v.rule =$ "$F \longrightarrow (R)$") return NFA($v.succ_2$));
if ($v.succ_1 \neq nil$ and $v.succ_2 = nil$) (* je jen jeden následník v *) return NFA($v.succ_1$);
}

Návrh bezkontextových gramatik; redukce

Nejprve jsme navrhli gramatiku, která řeší Příklad 1 z ukázkové druhé zápočtové písemky. Přitom jsme navrhli gramatiky G_1, G_2 pro jazyky L_1, L_2 a z nich jsme pak jednoduše získali gramatiku pro $L_1 \cdot L_2$ (jak je naznačeno v části 5.2., s. 166). Uvědomili jsme si, proč je obecně důležité zajistit, aby množiny neterminálů gramatik G_1 a G_2 byly disjunktní.

Pak jsme si znovu připomněli relaci $\Rightarrow$ na množině $(\Pi \cup \Sigma)^*$ pro danou bezkontextovou gramatiku $G = (\Pi, \Sigma, S, P)$. Také jsme si obecně připomněli *reflexivní a tranzitivní uzávěr* relace, konkrétně pak relaci $\Rightarrow^*$.

Víme tedy, že (pro danou G) je $\Rightarrow^*$ nejmenší množinou dvojic $(\alpha, \beta) \in (\Pi \cup \Sigma)^* \times (\Pi \cup \Sigma)^*$ splňující následující tři podmínky (pro každé $\alpha, \beta, \gamma \in (\Pi \cup \Sigma)^*$):

1. když $\alpha \Rightarrow \beta$, tak $\alpha \Rightarrow^* \beta$;
2. $\alpha \Rightarrow^* \alpha$;
3. když $\alpha \Rightarrow^* \beta$ a $\beta \Rightarrow^* \gamma$, tak $\alpha \Rightarrow^* \gamma$.

Na konkrétním příkladu

$$
\begin{aligned}
S &\longrightarrow aSb \,|\, aAbb \,|\, \varepsilon \\
A &\longrightarrow aAB \,|\, bB \\
B &\longrightarrow aAb \,|\, BB \\
C &\longrightarrow CC \,|\, cS
\end{aligned}
$$

jsme se pak zabývali redukcí bezkontextové gramatiky.

Přitom jsme obecně pro gramatiku $G = (\Pi, \Sigma, S, P)$ definovali množinu $\mathcal{D}_G$, značenou $\mathcal{D}$, když G je z kontextu jasná, jako nejmenší množinu splňující následující podmínky:

1. $S \in \mathcal{D}$;
2. když $X \in \mathcal{D}$ a $(X \longrightarrow \alpha Y \beta) \in P$, kde $Y \in \Pi$, tak $Y \in \mathcal{D}$.

Je vidět, že $\mathcal{D}$ obsahuje všechny dosažitelné neterminály, tedy takové, které se mohou objevit v nějakém odvození z počátečního S (byť to odvození nemusí třeba skončit terminálním slovem).

Množinu $\mathcal{T}_G$, stručněji $\mathcal{T}$, všech neterminálů, z nichž lze odvodit alespoň jedno terminální slovo, jsme definovali jako nejmenší množinu splňující následující podmínky:

1. když $(X \longrightarrow w) \in P$, kde $w \in \Sigma^*$, tak $X \in \mathcal{T}$;
2. když $(X \longrightarrow \alpha) \in P$, kde $\alpha \in (\Sigma \cup \mathcal{T})^*$, tak $X \in \mathcal{T}$.

Obě definice dávají očividné návody k algoritmické konstrukci příslušných množin; ty jsme si také ujasnili a provedli.

(Můžeme si všimnout, že v naší definici $\mathcal{T}$ je první podmínka zahrnuta v druhé a je tedy možné ji vypustit. Uvedli jsme ji proto, že explicitně vypichuje základní případ, tedy ty neterminály, u nichž se příslušnost k $\mathcal{T}$ zjistí hned v počáteční fázi algoritmu.)

Zjistili jsme, že pro zredukování gramatiky G je vhodné nejdříve zjistit množinu $\mathcal{T}_G$ a odstranit z G všechna pravidla, která obsahují alespoň jeden neterminál, který není v $\mathcal{T}_G$. Výsledná gramatika G' očividně generuje tentýž jazyk jako G.

> (Co to znamená, když nic nezbude, tedy ani S nepatří do $\mathcal{T}_G$? Samozřejmě $L(G) = \emptyset$.)

Pro G' nyní zkonstruujeme množinu $\mathcal{D}_{G'}$ a odstraníme všechna pravidla, která obsahují neterminál(y), jež nejsou v $\mathcal{D}_{G'}$. Výsledná gramatika G'' je již redukovaná. (Opačný postup, tj. odstranění nedosažitelných neterminálů a poté odstranění těch, které neodvodí terminální slovo, k redukované gramatice vést nemusí.)

Sekce pro hlubší zájemce – důkazy

Věnovali jsme se dvoucestným (či dvousměrným) konečným automatům. Takový (deterministický) automat, stručně 2-KA, je definován jako struktura

$$A = (Q, \Sigma, \#, \$, \delta, q_0, F)$$

kde Q, Σ, q_0, F mají stejný význam jako u standardních (jednosměrných) konečných automatů a $\#, \$ \notin \Sigma$ jsou speciální symboly označující levou a pravou zarážku. Přechodová funkce je nyní typu

$$\delta : (Q - F) \times (\Sigma \cup \{\#, \$\}) \to Q \times \{-1, +1\}\,.$$

Řídicí jednotka automatu A je čtecí hlavou spojena s páskou, na niž je na začátku zapsáno vstupní slovo $w \in \Sigma^*$ ohraničené na levé straně zarážkou # a na pravé straně zarážkou \$. Např. příslušná 'instrukce' $(q, a) \to (q', -1)$ znamená, že když automat A je ve stavu q a čte na pásce symbol a, pak přejde do stavu q' a posune hlavu o jedno políčko doleva ($+1$ znamená doprava). Na levé zarážce ovšem může pokračovat jen doprava, na pravé zarážce jen doleva. Další změna je v tom, že v přijímajícím stavu, prvku množiny F, výpočet končí (takový stav je opravdu *koncový*). Slovo je přijato, jestliže výpočet nad ním skončí (v přijímajícím stavu); není přijato, pokud je výpočet nekonečný. Z technických důvodů můžeme předpokládat, že do koncového stavu může automat přejít jen při čtení pravé zarážky \$.

Uvažujme nyní standardní konečný automat A a jazyk $L = \{u \mid uu \in L(A)\}$. Otázka je, jestli je L zaručeně regulární jazyk, tedy jestli existuje KA A' tak, že $L(A') = L$. To není na první pohled zřejmé. Jednoduché ale je zkonstruovat 2-KA A' tak, že $L(A') = L$. Takový A' dostane slovo $\#u\$$, odsimuluje A na u, po příjezdu na \$ si zapamatuje aktuální stav

automatu A, přejede na začátek a pokračuje v simulaci automatu A při druhém běhu na u.

Zamysleme se, jestli 2-KA umějí víc než KA, tedy jestli takové automaty rozpoznají i některé neregulární jazyky. Uvažujme tedy 2-KA $A = (Q, \Sigma, \#, \$, \delta, q_0, F)$ a podívejme se na množinu $\{w\backslash L(A) \mid w \in \Sigma^*\}$. Víme, že jestliže je tato množina konečná, pak je $L(A)$ regulární.

Podívejme se, jak vypadá výpočet A nad slovem $\#wu\$$. Jedna možnost je, že nikdy neopustí w; pak $w\backslash L(A) = \emptyset$. Jinak dojde k situaci, kdy výpočet poprvé opustí w vpravo. Stav, ve kterém výpočet poprvé vstoupí na u, záleží pouze na w, označme jej q_w. Když pak výpočet vstoupí na slovo w znovu (zprava) ve stavu q, pak buď už w neopustí, nebo se po nějaké době vrátí doprava na u ve stavu q'; stav q' je plně určen slovem w a oním vstupním stavem q. Je klíčové si uvědomit, že pro zjištění, zda slovo wu je automatem A přijato, nepotřebujeme znát kompletní slovo w, ale stačí nám informace označená $Beh(w)$ ('behaviour', tj. chování slova w), což je dvojice $Beh(w) = (q, f)$, kde $q \in Q \cup \{\perp\}$ a $f : Q \to Q \cup \{\perp\}$. Když je první komponenta rovna $\perp$, znamená to, že počáteční výpočet w nikdy neopustí (doprava); jinak tato komponenta udává výše zmíněný stav q_w. Druhá komponenta je funkcí; pro q určuje $f(q)$ onen návratový stav q', když výpočet vjede na slovo w zprava ve stavu q. Pochopitelně $f(q) = \perp$, jestliže po příjezdu na w zprava ve stavu q už slovo w nebude opuštěno (doprava).

Takže i jazyk $w\backslash L(A)$ je plně určen hodnotou $Beh(w)$. Jinými slovy, když $Beh(w) = Beh(w')$, tak $w\backslash L(A) = w'\backslash L(A)$. Ovšem funkce Beh nabývá jen konečně mnoha hodnot, takže množina $\{w\backslash L(A) \mid w \in \Sigma^*\}$ je opravdu konečná a jazyk $L(A)$ je tedy regulární. Dokázali jsme tedy:

Věta. Dvoucestné konečné automaty rozpoznávají právě regulární jazyky.

Partie textu k prostudování

Části 4.1. - 4.4. (bezkontextové gramatiky), část 5.1. (speciální bezkontextové gramatiky).

(Máte si udělat přinejmenším dobrou první představu a zamyslet se nad příklady, speciálně těmi plánovanými na cvičení, ať se můžete na cvičení aktivně účastnit a případné problémy si tam objasnit.)

Cvičení

Prezentace referátů

Referát č. 10

Vysvětlete pojem regulárních gramatik (RG) a jejich vztah ke konečným automatům. Na vhodných příkladech ilustrujte převody mezi (N)KA a RG a naopak. (Můžete vyjít z materiálu

`http://www.cs.vsb.cz/jancar/TEORET-INF/teoret-inf.pdf`

a/nebo z jiných zdrojů.)

Referát č. 11 (Redukce bezkontextové gramatiky)
Vysvětlete, co dělá algoritmus popsaný v sekci 3 publikace

> J. Esparza, P. Rossmanith, and S. Schwoon. A uniform framework for problems on context-free grammars. EATCS Bulletin, 72:169-177, October 2000,

která by měla být přístupná na
`http://www7.in.tum.de/um/bibdb/author-esparza.shtml`.
(Ilustrujte na vhodném příkladu.)
Pak stručně vysvětlete, jak je možné tento algoritmus využít při redukci gramatiky (tj. při "Identifying useless variables" na str. 8 zmíněné publikace).

Referát č. 12 (Chomského normální forma bezkontextových gramatik)
Při převodu bezkontextové gramatiky do Chomského normální formy (který je naznačen např. v textu
`http://www.cs.vsb.cz/jancar/TEORET-INF/teoret-inf.pdf`)
se po odstranění pravidel typu $A \to \varepsilon$ provádí krok, který odstraňuje pravidla typu $X \to Y$ (neterminál se přepisuje na neterminál). (Pak tedy dostaneme jen pravidla typu $A \to \alpha$, kde α je terminál nebo $|\alpha| \geq 2$, přičemž generovaný jazyk se nezměnil.)
Na vhodném příkladu ilustrujte algoritmus odstraňující ona pravidla $X \to Y$.

Příklady

Příklad 6.1

Připomeňme si gramatiku G

$$
\begin{aligned}
R &\longrightarrow T + R \mid T \\
T &\longrightarrow FT \mid F \\
F &\longrightarrow F^* \mid (R) \mid C \\
C &\longrightarrow a \mid b
\end{aligned}
$$

Ke každému slovu $w \in L(G)$ pochopitelně existuje příslušný derivační strom podle G (kde kořen je ohodnocen R a ohodnocení listů zleva doprava dává slovo w). Připomínáme, že strom je uspořádaný; každý vrchol má své následníky uspořádány „zleva doprava“. Tím je také indukováno uspořádání listů.
Snažte se co nejsrozumitelněji vyjádřit, co to vlastně znamená, že list ℓ_2 je vpravo od listu ℓ_1.
Řekneme, že (obecný) vrchol v *má napravo* list ℓ, jestliže ℓ je nejlevější z listů, které jsou vpravo od všech listů podstromu s kořenem v.
Nakreslete derivační strom pro nějaké (krátké) $w \in L(G)$, kde má nějaký vrchol ohodnocený T napravo list ohodnocený $+$.

Může mít v derivačním stromě pro (nějaké) $w \in L(G)$ vrchol ohodnocený T napravo list ohodnocený jinak než + ? Ukažte všechny možnosti. Může se také stát, že takový vrchol napravo žádný list nemá?
(Poznámka. Toto je příklad avízovaný v zápisu přednášky u Instrukce 9.)

Příklad 6.2

Pro bezkontextovou gramatiku $G = (\Pi, \Sigma, S, P)$ definujte množinu všech neterminálů, z nichž lze odvodit prázdné slovo (tedy ε), jako nejmenší množinu $\mathcal{E} \subseteq \Pi$ splňující ... (podobně jako u obdobných definic v přednášce).
Na základě této induktivní definice navrhněte algoritmus konstruující $\mathcal{E}$ a aplikujte jej na následující gramatiku G.

$$S \longrightarrow AB \,|\, \varepsilon$$
$$A \longrightarrow aAAb \,|\, BS \,|\, CA$$
$$B \longrightarrow BbA \,|\, CaC \,|\, \varepsilon$$
$$C \longrightarrow aBB \,|\, bS$$

Teď bychom chtěli zkonstruovat gramatiku G', která nebude obsahovat tzv. ε-pravidla, tedy pravidla s pravou stranou ε, ale bude generovat stejný jazyk jako G, s případnou výjimkou ε (tedy $L(G') = L(G) - \{\varepsilon\}$).
Můžeme prostě ε-pravidla z G vypustit a prohlásit výslednou gramatiku za kýženou G' ? Vysvětlete, proč ne.
Zkuste přijít na (obecně platný) postup, jak G' sestrojit, když známe množinu $\mathcal{E}$.

Příklad 6.3

Navrhněte bezkontextovou gramatiku generující jazyk všech palindromů v abecedě $\{a, b\}$, jejichž délka je násobkem tří.
(Jedná se tedy o jazyk $L = \{\, w \in \{a,b\}^* \mid w = w^R \text{ a } (|w| \bmod 3) = 0 \,\}$.)
Zkuste nejprve najít gramatiku používající jediný neterminál. Poté popřemýšlejte, jestli použitím více neterminálů dokážete počet potřebných pravidel zmenšit.
(Nakonec porovnejte s řešením Cvičení 4.13. na s. 137.)