VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

DIPLOMOVÁ PRÁCE

2002 Martin Kot

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Řízené gramatiky

2002 Martin Kot

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne . . . .

Za poskytnutou pomoc, podporu a podnětné připomínky děkuji vedoucímu této diplomové práce, doc. Jaroslavu Marklovi.

Abstrakt

V teorii formálních jazyků se většinou používají klasické gramatiky. Dělíme je podle Chomského hierarchie. V této práci jsou představeny bezkontextové řízené gramatiky, které jsou často vhodnější. Omezují i užití pravidel a ne jen jejich tvar. Různé typy řízení derivace jsou zde srozumitelně vysvětleny. Neformální popis je pro přesnost doplněn definicemi. Pro snadné pochopení jsou zde komentované příklady. Řízené gramatiky jsou porovnány s klasickými i mezi sebou podle generativní síly, přirozenosti návrhu a složitosti algoritmu derivace. Na jednoduchém programovacím jazyku jsou ukázány výhody řízené derivace. Teorie je doplněna o softwarový simulátor derivace podle programované gramatiky.

Klíčová slova

gramatika, řízená gramatika, programovaná gramatika, maticová gramatika, regulárně řízená gramatika, neuspořádaná vektorová gramatika, podmíněná gramatika, polopodmíněná gramatika, gramatika s náhodným kontextem, uspořádaná gramatika, indická paralelní gramatika, k-gramatika, rozptýlená kontextová gramatika, neuspořádaná rozptýlená kontextová gramatika, k-jednoduchá maticová gramatika, řízená derivace

Používané zkratky a symboly

C ..... Rodina podmíněných gramatik
CF ..... Rodina bezkontextových gramatik
CS ..... Rodina kontextových gramatik
IP ..... Rodina indických paralelních gramatik
$\mathfrak{L}(X)$ .... Rodina jazyků příslušných rodině gramatik X
$L(G)$ ..... Jazyk generovaný gramatikou G
M ..... Rodina maticových gramatik
M_{ac} ..... Rodina maticových gramatik s testováním výskytu
N ..... Množina neterminálních symbolů
P ..... Rodina programovaných gramatik
P_{ac} ..... Rodina programovaných gramatik s testováním výskytu
O ..... Rodina uspořádaných gramatik
RC_{ac} ..... Rodina gramatik s náhodným kontextem
RE ..... Rodina obecných generativních gramatik
SC ..... Rodina rozptýlených kontextových gramatik
SM ..... Rodina k-jednoduchých maticových gramatik pro všechna přirozená k
T ..... Množina terminálních symbolů
V_G ..... Množina všech symbolů (terminálních i neterminálních)
kG ..... Rodina k-gramatik
rC ..... Rodina regulárně řízených gramatik
sC ..... Rodina polopodmíněných gramatik
uSC ..... Rodina neuspořádaných rozptýlených kontextových gramatik
uV ..... Rodina neuspořádaných vektorových gramatik
λ ..... Prázdné slovo
λX ..... Rodina gramatik typu X s vypouštěcími pravidly
$|w|$ ..... Délka slova w
$|w|_M$ ..... Počet výskytů symbolů z množiny M ve slově w

Česko-anglický slovník pojmů

bezkontextová gramatika	context-free grammar
gramatika	grammar
gramatika s náhodným kontextem	random context grammar
indická paralelní gramatika	indian parallel grammar
k-gramatika	k-grammar
k-jednoduchá maticová gramatika	k-simple matrix grammar
kontextová gramatika	context-sensitive grammar
maticová gramatika	matrix grammar
neterminál	nonterminal
neuspořádaná rozptýlená kontextová gramatika	unordered scattered context grammar
neuspořádaná vektorová gramatika	unordered vector grammar
obecná generativní gramatika	arbitrary phrase-structure grammar
podmíněná gramatika	conditional grammar
pole neúspěchu	failure field
pole úspěchu	success field
polopodmíněná gramatika	semi-conditional grammar
povolený kontext	permitted context
pravidlo	production, rule
programovaná gramatika	programmed grammar
regulárně řízená gramatika	regularly controlled grammar
rodina gramatik	familly of grammars
rodina jazyků	familly of languages
rozptýlená kontextová gramatika	scattered context grammar
řízená derivace	controlled derivation
řízená gramatika	grammar with controlled derivations
terminál	terminal
testování výskytu	appearance checking
uspořádaná gramatika	ordered grammar
vypouštěcí pravidlo	erasing rule
zakázaný kontext	forbidden context

Obsah

1. Úvod

V teorii formálních jazyků se nejčastěji na rozdělení jazyků do kategorií používá Chomského hierarchie. Ta zavádí 4 skupiny jazyků podle omezení kladených na tvar pravidel gramatik, generujících tyto jazyky. Programátoři používají z této hierarchie nejčastěji bezkontextové gramatiky, například pro zadání jazyka pro generátor překladače. Problémem je, že velká část programovacích jazyků obsahuje některé prvky, které tímto typem gramatiky nelze popsat. Jedná se zejména o nutnost deklarace identifikátorů před jejich použitím. V již zmíněných překladačích se na vyřešení deklarací zavádějí speciální konstrukce, například tabulky symbolů. Obdobně lingvisté považují přirozené jazyky sice za kontextové, ale velká část by se dala popsat bezkontextovou gramatikou. Jen několik jazykových konstrukcí řadí přirozené jazyky do skupiny kontextových.

Nabízela by se možnost využívat kontextové gramatiky. [IAT–79] uvádí, že téměř všechny jazyky, které můžeme vymyslet, jsou kontextové. Jediný známý důkaz o existenci jazyků nepatřících mezi kontextové je založen na diagonalizaci. Kontextové jazyky sice mají velkou generativní sílu, ale jejich velkou nevýhodou je podle [HFL–96] například exponenciální složitost všech známých algoritmů rozhodujících o příslušnosti slova do jazyka. Navíc problém prázdnosti jazyka je nerozhodnutelný.

Tato práce se zaměří na jinou možnost vyřešení některých kontextových vlastností jazyků. Rozšíření nespočívá ve změně tvaru pravidel, ale v zavedení nějakého mechanizmu, který povolí jen některé derivace. Získáme tak podmnožinu jazyka generovaného gramatikou běžným způsobem. Tato podmnožina může být kontextovým jazykem. Snahou je zmíněné mechanizmy volit tak, aby zůstalo co nejvíc dobrých vlastností bezkontextových gramatik, například rozhodnutelnost příslušnosti slova k jazyku algoritmem s polynomiální složitostí. Současně si přejeme zvětšit co nejvíce generativní schopnost gramatik. V práci budou uvedeny příklady takto doplněných bezkontextových gramatik, které budou generovat jazyky typu 0 Chomského hierarchie. Protože přidaný mechanismus spočívá ve výběru jen některých možných derivací, neboli řízením derivace, nazývají se takto doplněné gramatiky řízené (resp. gramatiky s řízenými derivacemi, gramatiky s řízeným přepisováním).

Cílem práce je představit čtenáři různé typy řízených gramatik. Abychom ukázali větší generativní sílu proti klasickým bezkontextovým gramatikám, ukážeme si několik příkladů řízených gramatik pro kontextové jazyky. Příklady budou důkladně komentované, aby čtenář snadno pochopil způsob, jakým zajišťují generování požadovaného jazyka. Porovnáme výhody a nevýhody jednotlivých typů řízení a pokusíme se najít ten nejvhodnější pro praktické užití. Vybranou řízenou gramatiku využijeme i pro popis složitějšího a praktičtějšího jazyka. Na tomto jazyce si ukážeme výhody řízených gramatik proti klasickým.

Pro upřesnění notace používané v celé práci si uvedeme definici obecné generativní gramatiky i zmiňované Chomského hierarchie.

Definice 1.1 *Generativní gramatika je čtveřice $G = (N, T, P, S)$, kde*

- *N je konečná množina neterminálních symbolů (neterminálů),*

- *T je konečná množina terminálních symbolů (terminálů), $T \cap N = \phi$*

- *P je konečná množina pravidel tvaru $x \to y$, kde*

$$x \in (N \cup T)^* N (N \cup T)^* \text{ a } y \in (N \cup T)^*,$$

- *$S \in N$ je počáteční neterminál.*

Pro $u, v \in (N \cup T)^*$ řekneme, že u se přímo přepíše na v (podle gramatiky G) a značíme $u \Rightarrow_G v$ nebo $u \Rightarrow v$, je-li gramatika zřejmá z kontextu, jestliže existují slova w_1, w_2, x, y taková, že $u = w_1 x w_2, v = w_1 y w_2$ a $x \to y \in P$.

Říkáme, že u se přepíše na v a značíme $u \Rightarrow^* v$, jestliže existuje posloupnost $w_0, w_1, \ldots, w_n$ slov z $(T \cup N)^*$ pro nějaké $n \geq 0$ takové, že $u = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n = v$. Tato posloupnost se nazývá derivací nebo odvozením délky n slova v ze slova u.

Jazyk $L(G)$ generovaný gramatikou G je definován jako $L(G) = \{w \in T^* | S \Rightarrow^* w\}$.

Gramatiky G_1, G_2 nazýváme ekvivalentní právě tehdy, když $L(G_1) = L(G_2)$.

Definice 1.2 *Chomského hierarchie gramatik a jimi generovaných jazyků je definována takto:*

- *Obecná generativní gramatika definovaná v definici 1.1 je typu 0.*

- *G je gramatika typu 1, neboli kontextová gramatika, jestliže každé pravidlo v P je ve tvaru $uXv \to uwv$, kde $|w| \geq 1$, $u, v, w \in (T \cup N)^*, X \in N$. Výjimkou může být pouze pravidlo $S \to \lambda$, ale S se potom nesmí vyskytovat na pravé straně žádného pravidla.*

- *G je gramatika typu 2, neboli bezkontextová, jestliže každé pravidlo v množině P je tvaru $X \to u$, $u \in (T \cup N)^*$, $X \in N$.*

- *G je gramatika typu 3, neboli regulární gramatika právě tehdy, když je každé pravidlo z P tvaru $X \to wY$ nebo $X \to w$, kde $w \in T^*$, $X, Y \in N$.*

  Jazyk L je typu i, kde $i = 0, 1, 2, 3$, jestliže existuje gramatika typu i, která jej generuje. Jazyk nazýváme kontextový, resp. bezkontextový, regulární, jestliže jej generuje nějaká kontextová, resp. bezkontextová, regulární gramatika.

Předpokládáme, že pojmy abeceda, jazyk, slovo, podslovo, symbol abecedy, terminální a neterminální symbol a další základní pojmy teorie formálních jazyků jsou čtenáři známé a jejich definice zde neuvádíme. Je možné je najít v různých publikacích zabývajících se formálními jazyky, například [Sal–73].

2. Typy řízených gramatik

Řízené gramatiky vycházejí z klasických. Mají stejná pravidla, množiny neterminálů a terminálů a počáteční neterminál. Obsahují však navíc nějaký mechanizmus, který povolí jen některé derivace ze všech možných. V každém kroku derivace se mohou používat pravidla jen z podmnožiny množiny všech pravidel. Tato podmnožina není dána jen tvarem pravidel, jak je to v klasických gramatikách, ale i oním přidaným mechanizmem. Omezením počtu aplikovatelných pravidel řídíme derivaci, proto se gramatiky nazývají řízené.

Přidaný mechanizmus řídící derivaci může být založen na různých algoritmech a principech. Například aplikace pravidla v jednom kroku určí použitelná pravidla pro další krok nebo množina aplikovatelných pravidel závisí na aktuální větné formě. Proto i typů řízených gramatik je hodně. Některé z nich si zde uvedeme.

Z Chomského hierarchie se často používají bezkontextové gramatiky. Jejich výhodami jsou snadný návrh a polynomiální algoritmus určení příslušnosti slova k jazyku. Nevýhodou je malá generativní síla. Doplníme-li je o řízení, můžeme získat generativní sílu na úrovni obecných gramatik. Proto nepotřebujeme používat kontextová nebo obecná pravidla. Generativní síla by se již nezměnila a ztratily by se výhody bezkontextových gramatik. V dalším textu budeme tedy uvažovat pouze bezkontextové řízené gramatiky.

Všechny níže uvedené typy gramatik jsou popsány v [HFL–96]. Odtud jsou převzaty i jejich definice.

2.1 Předepsané sekvence

Množinu aplikovatelných pravidel můžeme určovat podle pravidla použitého v minulém kroku nebo krocích. Při tvorbě konkrétní gramatiky přiřadíme každému pravidlu jiná, která mohou být užita po něm. Způsobem, jakým se to provede, se liší níže popsané typy gramatik.

2.1.1 Maticová gramatika

V maticových gramatikách je dáno několik samostatných sekvencí pravidel. Při derivaci vždy některou z nich vybereme. Všechna její pravidla se automaticky aplikují v daném pořadí. Proto se musí vybrat taková posloupnost, aby byla aplikovatelná celá. Jinak by derivace nevedla k terminálnímu slovu. Tím, že si vynutíme po použití prvního pravidla sekvence i aplikaci ostatních, můžeme zajistit generování různých jazykových konstrukcí, které by bezkontextovými gramatikami bez řízení generovat nešly. Musíme jen vhodně seřadit pravidla do sekvencí při návrhu.

Formálně jsou sekvence reprezentovány vektory pravidel. Jejich množina nahrazuje ve specifikaci maticových gramatik množinu pravidel z bezkontextových gramatik. Nejprve lze aplikovat první pravidlo libovolného vektoru. V následujících krocích je nutné postupně užít všechna pra-

vidla vektoru v daném pořadí. Až po použití všech je možno aplikovat opět první pravidlo libovolného vektoru. Zmíněné vektory pravidel se nazývají matice.

Definice 2.1 *a) Maticová gramatika* (matrix grammar) *je čtveřice* $G = (N, T, M, S)$*, kde*

- N, T, S *jsou definovány stejně jako v bezkontextových gramatikách,*
- $M = \{m_1, m_2, \ldots, m_n\}$, $n \geq 1$*, je konečná množina sekvencí*

$$m_i = (p_{i_1}, \ldots, p_{i_{k(i)}}),\ k(i) \geq 1,\ 1 \leq i \leq n,$$

kde všechna p_{i_j}, $1 \leq i \leq n$, $1 \leq j \leq k(i)$*, jsou bezkontextová pravidla,*

b) Pro $m_i, 1 \leq i \leq n$*, a* $x, y \in V_G^*$*, definujeme* $x \Rightarrow_{m_i} y$ *jako*

$$x = x_0 \Rightarrow_{p_{i_1}} x_1 \Rightarrow_{p_{i_2}} x_2 \Rightarrow_{p_{i_3}} \ldots \Rightarrow_{p_{i_{k(i)}}} = y,$$

kde $m_i = (p_{i_1}, \ldots, p_{i_{k(i)}}) \in M$

c) Jazyk $L(G)$ *generovaný gramatikou* G *je definován jako množina všech slov* $w \in T^*$ *takových, že existuje derivace*

$$S \Rightarrow_{m_{j_i}} y_1 \Rightarrow_{m_{j_2}} y_2 \Rightarrow_{m_{j_3}} \ldots \Rightarrow_{m_{j_s}} w,$$

pro nějaké $s \geq 1,\ 1 \leq j_i \leq n,\ 1 \leq i \leq s$.

Symbolem λM označujeme rodinu maticových gramatik. M je rodina maticových gramatik bez vypouštěcích pravidel.

Pokud se v matici nachází pravidlo, jehož levá strana se v době nutnosti aplikace nevyskytuje v derivované větné formě, nelze použít celou matici. Užití pravidla můžeme definovat i v širším smyslu. Pokud se jeho levá strana nevyskytuje v derivovaném slově, nechá se slovo beze změny a mluvíme o aplikaci pravidla ve smyslu testování výskytu. Užití v módu testování výskytu je tedy zjištění, že pravidlo v běžném smyslu použít nelze. V dalším kroku se aplikuje následující pravidlo matice na nezměněnou větnou formu. Tato aplikace může být buď klasická nebo opět v módu testování výskytu.

Uvedená možnost užití pravidel ve smyslu testování výskytu je u maticových gramatik omezena jen na některá pravidla. Ta jsou v množině F, která je pátým prvkem gramatiky. Původní čtveřice definující gramatiku je tak rozšířena na pětici. Je možné mít stejné pravidlo v maticích m_1 a m_2, přičemž výskyt v matici m_1 patří do množiny F a výskyt v m_2 nepatří do F. Následující definice zahrnuje i maticové gramatiky bez testování výskytu a tak nahrazuje definici 2.1.

Definice 2.2 *a) Maticová gramatika s testováním výskytu* (matrix grammar with appearance checking) *je pětice* $G = (N, T, M, S, F)$*, kde*

- N, T, S, M *jsou definovány stejně jako v maticové gramatice v definici 2.1,*

- *F je podmnožina množiny všech pravidel v maticích, tzn.*

$$F \subseteq \{p_{i_j} \mid 1 \leq i \leq n, 1 \leq j \leq k(i)\}.$$

b) Pro pravidlo $p = A \to w \in P$ a $x, y \in V_G^*$ definujeme aplikaci

$$x \Rightarrow_p^{ac} y$$

pravidla p v módu testování výskytu jako

$$x = x_1 A x_2 \text{ a } y = x_1 w x_2,\ x_1, x_2 \in V_G^*$$

nebo

$$x = y,\ A \text{ není obsaženo v } x,\ p \in F.$$

c) Pro $m_i,\ 1 \leq i \leq n$, a $x, y \in V_G^*$, definujeme $x \Rightarrow_{m_i}^{ac} y$ jako

$$x = x_0 \Rightarrow_{p_{i_1}}^{ac} x_1 \Rightarrow_{p_{i_2}}^{ac} x_2 \Rightarrow_{p_{i_3}}^{ac} \ldots \Rightarrow_{p_{i_{k(i)}}}^{ac} = y,$$

d) Jazyk $L(G)$ generovaný maticovou gramatikou G s testováním výskytu je definován jako množina všech slov $w \in T^*$ takových, že existuje derivace

$$S \Rightarrow_{m_{j_1}}^{ac} y_1 \Rightarrow_{m_{j_2}}^{ac} y_2 \Rightarrow_{m_{j_3}}^{ac} \ldots \Rightarrow_{m_{j_s}}^{ac} w,$$

pro nějaké $s \geq 1,\ 1 \leq j_i \leq n,\ 1 \leq i \leq s$.

e) G je maticová gramatika bez testování výskytu právě tehdy, když $F = \phi$.

Často se ve značení $x \Rightarrow^{ac} y$ horní index vynechává a píše se pouze $x \Rightarrow y$. Ze specifikace gramatiky je zřejmé, že se jedná o použití matic v módu testování výskytu a není nutné to uvádět.

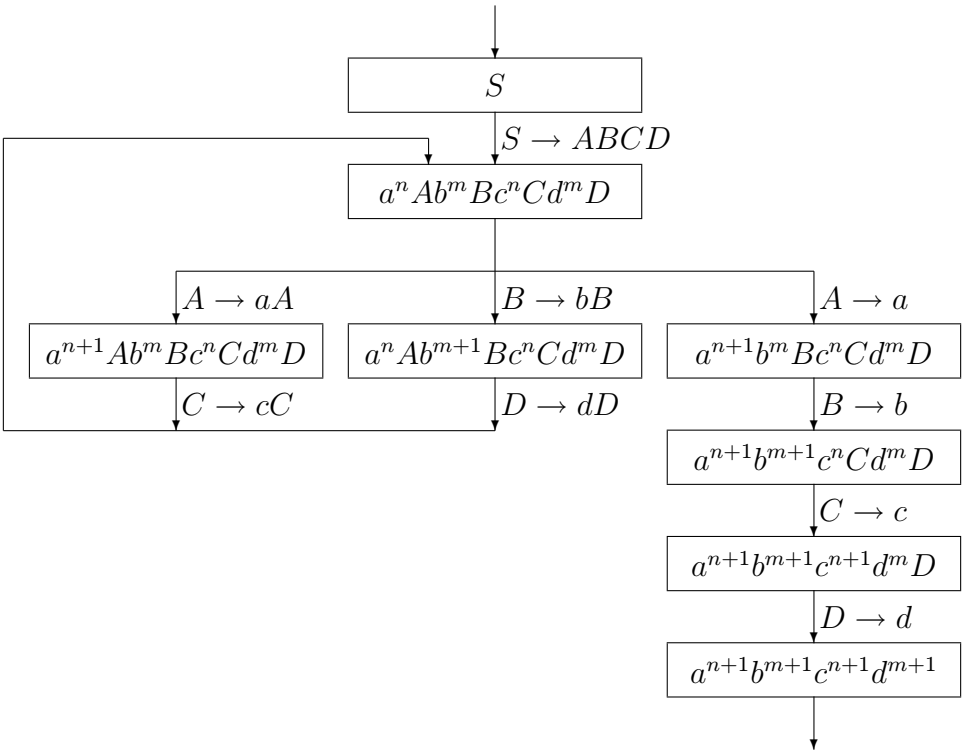
Symbolem λM_{ac} označujeme rodinu maticových gramatik s testováním výskytu. M_{ac} je rodina maticových gramatik s testováním výskytu bez vypouštěcích pravidel.

Jestliže jsou všechny matice maticové gramatiky jednoprvkové, jedná se o běžnou bezkontextovou gramatiku. Tedy ke každé bezkontextové gramatice existuje maticová gramatika. Z toho plyne $\mathcal{L}(CF) \subseteq \mathcal{L}(M)$.

Příklad 2.1 $L(G) = \{a^n b^m c^n d^m | n, m \geq 1\}$

$G = (\{S, A, B, C, D\}, \{a, b, c, d\}, \{m_1, \ldots, m_4\}, S, \phi)$
$m_1 = (A \to aA,\ C \to cC)$ $\quad m_2 = (A \to a,\ B \to b,\ C \to c,\ D \to d)$
$m_3 = (S \to ABCD)$ $\quad m_4 = (B \to bB,\ D \to dD)$

Na obrázku 2.1 je schématicky znázorněn průběh derivace. Jen v jednom bodě je nedeterminismus při volbě pravidla. Při větné formě $a^n A b^m B c^n C d^m D$ se volí matice podle toho, jestli si přejeme přidat další a a c nebo b a d nebo skončit derivaci odstraněním všech neterminálů. Jinak se pravidla aplikují automaticky, protože vybraná matice musí být postupně použita celá. Na začátku derivace po aplikaci $S \to ABCD$ je $n = 0,\ m = 0$. Potom se postupně po použití některé z matic n nebo m zvětší o 1.



Obrázek 2.1: Schéma průběhu derivace pro $a^n b^m c^n d^m,\ n, m \geq 1$ podle maticové gramatiky

Příklad 2.2 $L(G) = \{a^n b^m a^n | 0 \leq m \leq n\}$

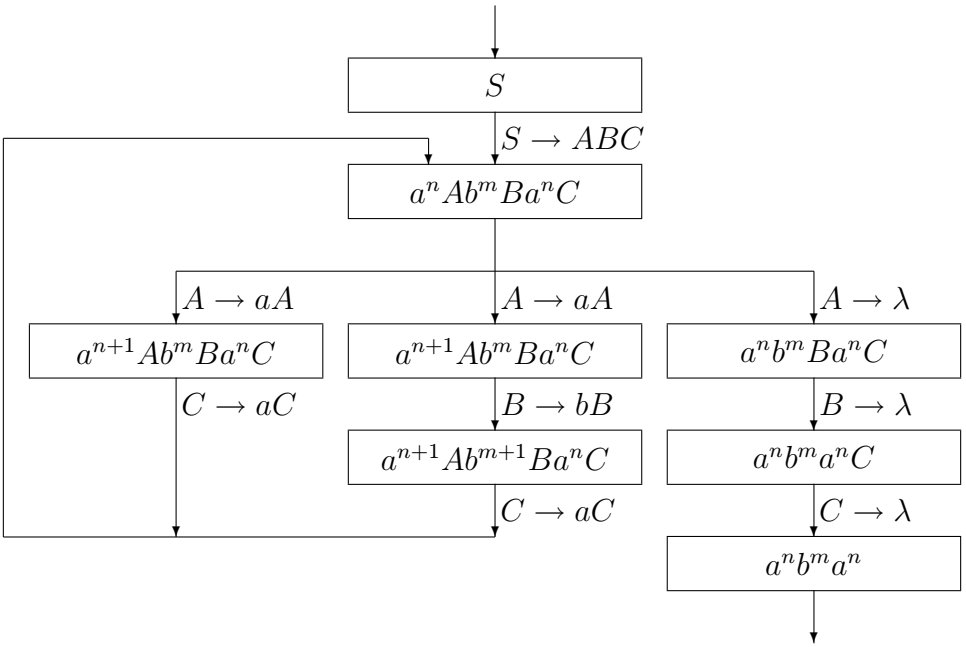
$G = (\{S, A, B, C\}, \{a, b\}, \{m_1, m_2, m_3, m_4\}, S, R, \phi)$
$m_1 = (S \to ABC)$ $\quad m_2 = (A \to aA,\ C \to aC)$
$m_3 = (A \to aA,\ B \to bB,\ C \to aC)$ $\quad m_4 = (A \to \lambda,\ B \to \lambda,\ C \to \lambda)$

Na obrázku 2.2 je opět schématicky zobrazen průběh derivace. Začíná se aplikací pravidla $S \to ABC$. Na začátku tedy jsou $n, m = 0$. Potom nastává okamžik rozhodnutí. Můžeme přidat dvojici symbolů a a současně jedno b, přidat pouze dvojici a, nebo můžeme derivaci ukončit přepsáním všech neterminálů pomocí vypouštěcích pravidel. Protože větev s vypouštěcími pravidly může být využita, aniž by se vygenerovala nějaká a nebo b, generuje gramatika i prázdné slovo. b se přidává jen současně s dvojicí a. Tím je zaručeno $m \leq n$.

Gramatiky pro jazyky $L = \{a^n b^n c^n | n \geq 1\}$ a $L = \{a^{2^m} | m \geq 1\}$ lze nalézt v [Sal–73] a pro $L = \{ww | w \in \{a, b\}^+\}$ v [HFL–96].

2.1.2 Regulárně řízená gramatika

Regulárně řízené gramatiky využívají jiný mechanizmus určení pořadí pravidel. Nejsou dány jen krátké sekvence, jakými byly matice v maticových gramatikách, ale celá derivace. Nejde



Obrázek 2.2: Schéma průběhu derivace pro $a^n b^m a^n,\ 0 \leq m \leq n$ podle maticové gramatiky

ovšem o určení jednoznačné, protože by bylo možné generovat pouze jedno slovo. Pro umožnění opakování a volby z variant se zavádí regulární množina povolených posloupností pravidel. Ta je součástí specifikace konkrétní regulárně řízené gramatiky. Každá sekvence pravidel vedoucí k terminálnímu slovu z jazyka do ní musí patřit.

Mechanizmus regulárních gramatik můžeme ekvivalentně chápat i následovně. Každému pravidlu je přiřazeno jednoznačné označení. Je definován regulární jazyk nad množinou označení těchto pravidel. Označení pravidel užitých v derivaci tvoří slovo a to musí patřit do tohoto jazyka. Nejčastěji je jazyk určen regulárním výrazem. Při derivaci musíme v každém kroku vybrat pravidlo pro aplikaci tak, aby výsledná sekvence použitých pravidel mohla vyhovovat regulárnímu výrazu.

Obdobně jako maticové umožňují i regulárně řízené gramatiky užití pravidel v módu testování výskytu. Opět je tato možnost omezena jen na pravidla uvedená v množině, která je součástí specifikace gramatiky. Pokud je množina prázdná, jedná se o gramatiku bez testování výskytu. Následující definice zahrnuje obě možnosti aplikace pravidel.

Mohlo by se zdát, že ekvivalentním krokem k povolení testování výskytu zařazením pravidla do množiny F je posloupnost pravidel s volitelným užitím tohoto pravidla. Je v tom však rozdíl. Pokud se neterminál z levé strany pravidla ve větné formě vyskytuje, je při testování výskytu vždy přepsán. Při druhé zmiňované variantě by mohlo být pravidlo vynecháno, i když je levá strana součástí větné formy, a neterminál by zůstal nepřepsán.

Definice 2.3 *a) Regulárně řízená gramatika s testováním výskytu* (regularly controlled grammar with appearance checking) *je šestice* $G = (N, T, P, S, R, F)$*, kde*

- N, T, P, S *jsou definovány stejně jako v bezkontextových gramatikách,*
- R *je regulární množina nad P,*
- F *je podmnožina P.*

b) Pro pravidlo $p = A \to w \in P$ *a* $x, y \in V_G^*$*, definujeme aplikaci*

$$x \Rightarrow_p^{ac} y$$

pravidla p v módu testování výskytu jako

$$x = x_1 A x_2 \quad a \quad y = x_1 w x_2$$

nebo

$$x = y, \quad A \text{ není obsaženo v } x, \; p \in F.$$

c) Jazyk $L(G)$ *generovaný regulárně řízenou gramatikou* G *s testováním výskytu se skládá ze všech slov* $w \in T^*$ *takových, že existuje derivace*

$$S \Rightarrow_{p_1}^{ac} w_1 \Rightarrow_{p_2}^{ac} w_2 \Rightarrow_{p_3}^{ac} \ldots \Rightarrow_{p_n}^{ac} w_n = w, \text{ kde } p_1 p_2 \ldots p_n \in R$$

d) G *je regulárně řízená gramatika bez testování výskytu právě tehdy, když* $F = \phi$*.*

Regulárně řízené gramatiky jsou zřejmě nadmnožinou bezkontextových gramatik. Pokud je $F = \phi$ a $R = P^*$ není možné používat pravidla v módu testování výskytu a jsou povolené všechny sekvence pravidel při derivaci. Jedná se tedy o bezkontextovou gramatiku.

Rodinu regulárně řízených gramatik označujeme λrC_{ac}. Pokud se jedná o gramatiku bez vypouštějících pravidel, vynecháme v označení λ. Pokud není povoleno testování výskytu, vynecháme index ac.

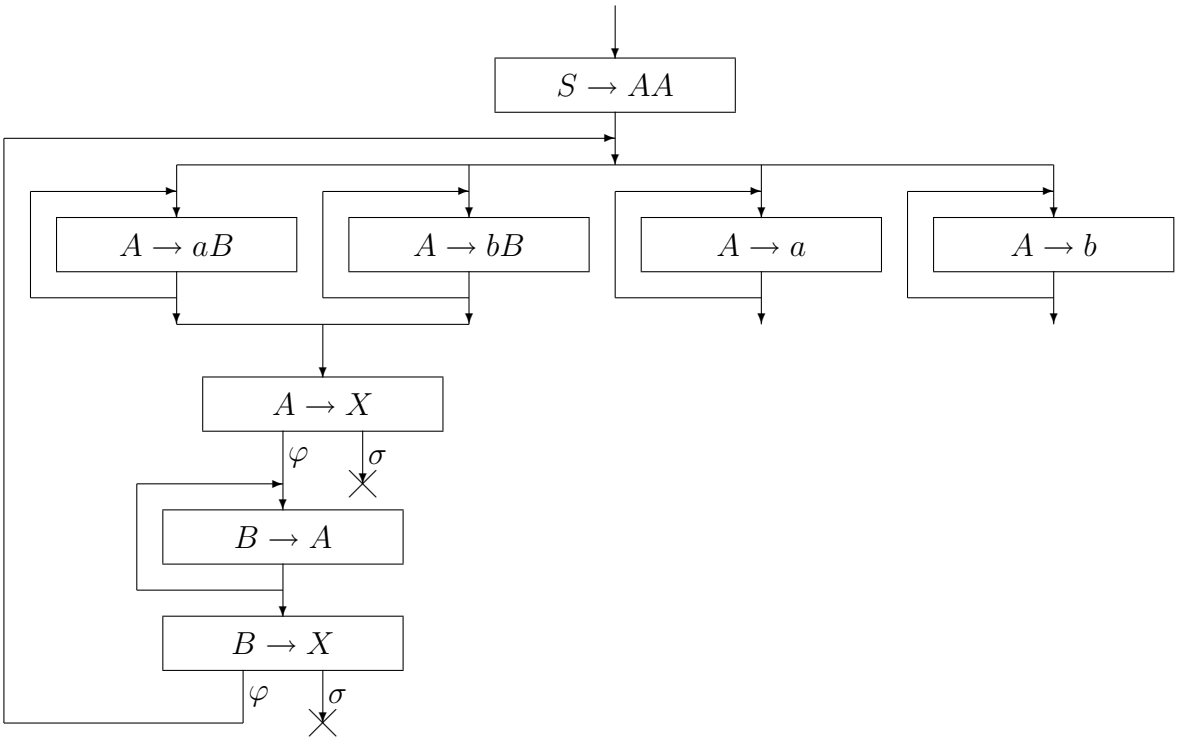
Příklad 2.3 $L(G) = \{ww | w \in \{a, b\}^+\}$

$G = (\{S, A, B, X\}, \{a, b\}, \{p_1, p_2, \ldots, p_8\}, S, R, \{p_6, p_8\})$
$R = \{p_1((p_2^* + p_3^*)p_6 p_7^* p_8)^*(p_4^* + p_5^*)\}$

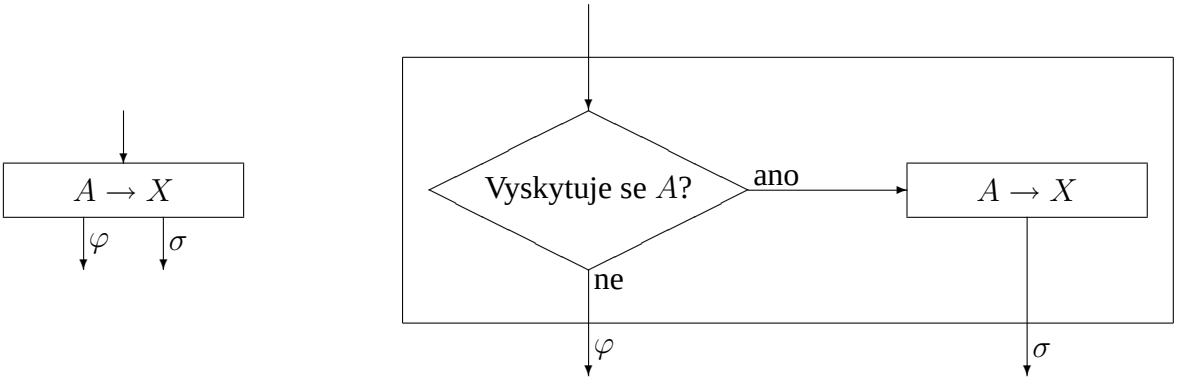
$$p_1 = S \to AA \quad p_2 = A \to aB \quad p_3 = A \to bB \quad p_4 = A \to a$$
$$p_5 = A \to b \quad p_6 = A \to X \quad p_7 = B \to A \quad p_8 = B \to X$$

Na obrázku 2.3 je vývojový diagram znázorňující, jak jsou postupně aplikována pravidla při derivaci. Většina pravidel neumožňuje použití v módu testování výskytu. Šipka vycházející z nich proto znamená klasickou aplikaci. Z pravidel patřících do množiny F vycházejí dvě šipky. Označení φ znamená pokračování po použití v módu testování výskytu, σ po aplikaci klasické. Na obrázku 2.4 je znázorněno, jak by mohl být obdélník znázorňující pravidlo z F (na obrázku vlevo) chápán vnitřně (vpravo).

V prvním kroku se zavede dvojice neterminálů A. Ty postupně budou tvořit dvě stejná slova. Jeden ze symbolů A můžeme přepsat na terminál a nebo b. Potom je možné užít jen stejné pravidlo i na druhé A. Tím derivace končí.



Obrázek 2.3: Použití pravidel při generování $ww,\ w \in \{a, b\}^*$ podle regulárně řízené gramatiky



Obrázek 2.4: Význam schématického znázornění pravidla s možností použití ve smyslu testování výskytu

Před ukončením derivace se také může A přepsat podle jednoho ze dvou pravidel s B v pravé straně. Zvolené pravidlo potom musíme užít i na druhé A. Kdyby totiž A zůstalo nepřepsáno, další krok ho převede na X. Symbol X není jak odstranit, protože se nevyskytuje na levé straně žádného pravidla, a derivace je zablokovaná (křížek v obrázku). Jsou-li ve větné formě jen terminály a dvě B, pravidlo $A \to X$ se použije ve smyslu testování výskytu. Potom se

začnou B přepisovat na A. Aplikace pravidla na obě B je opět zajištěna případným přepsáním na neterminál X a zablokováním derivace. Jsou-li ve větné formě jen symboly A a terminály, volí se další pravidlo zase ze čtyř možností. Protože se vždy oba symboly A přepisují stejně, je zajištěno, že vygenerují stejná slova.

Gramatika pro daný jazyk by mohla být jednodušší. Regulární výraz

$$p_1(p_2p_2 + p_3p_3)p_7p_7(p_4p_4 + p_5p_5)$$

při zachování značení pravidel zajistí generování stejného jazyka. Pravidla p_6 a p_8 potom již nejsou potřeba. Původní gramatika může však být jednodušeji změněna tak, aby generovala jazyk s jiným počtem opakování stejného slova. Změnilo by se pouze pravidlo p_1. Navíc by se dala zakomponovat jako část větší gramatiky, ve které se počet opakování stejných slov liší v různých derivacích. Zjednodušená verze je omezená na právě dvě opakování slova. Při změně tohoto počtu by se musel měnit i regulární výraz. Výhodou je ale determinističtější průběh derivace. Přepíše vždy oba výskyty A automaticky stejným pravidlem. Ve variantě s testováním výskytu se musí po aplikaci pravidla na první A rozhodovat, jestli se nemá použít $A \to X$. Přitom musíme nahlížet dopředu, abychom zjistili, že při jeho užití by se derivace zablokovala.

Příklad 2.4 $L(G) = \{a^n b^m c^n d^m | n, m \geq 1\}$

$G = (\{S, A, B, C, D\}, \{a, b, c, d\}, \{p_1, p_2, \ldots, p_9\}, S, R, \phi)$
$R = \{p_1(p_2p_4)^*(p_3p_5)^*p_6p_7p_8p_9\}$

$p_1 = S \to ABCD$	$p_2 = A \to aA$	$p_3 = B \to bB$
$p_4 = C \to cC$	$p_5 = D \to dD$	$p_6 = A \to a$
$p_7 = B \to b$	$p_8 = C \to c$	$p_9 = D \to d$

Na obrázku 2.5 je schéma průběhu derivace k příkladu 2.4. I když jsou pravidla podobná jako v příkladu 2.1, je postup derivace trochu odlišný. Zvolený regulární výraz nám zajistí, že nejprve se vytvoří $n-1$ symbolů a a c a potom $m-1$ terminálů b a d. Nakonec jsou všechny neterminály přepsány na poslední výskyt každého terminálu.

V maticové gramatice není určeno, v jakém pořadí se matice používají, a je možné střídavě přidávat dvojici a a c a dvojici b a d. Tento rozdíl se projeví při rozhodování v bodech neurčitosti v průběhu derivace. V regulárně řízené gramatice se rozhoduje, jestli se pokračuje v přidávání a a c (resp. b a d), nebo se přejde na druhou dvojici (resp. přepis neterminálů na terminály a konec derivace). Přidávají-li se b a d, nelze se již vrátit k a a c. V maticové se v každém cyklu volí přidání libovolné dvojice nebo konec derivace.

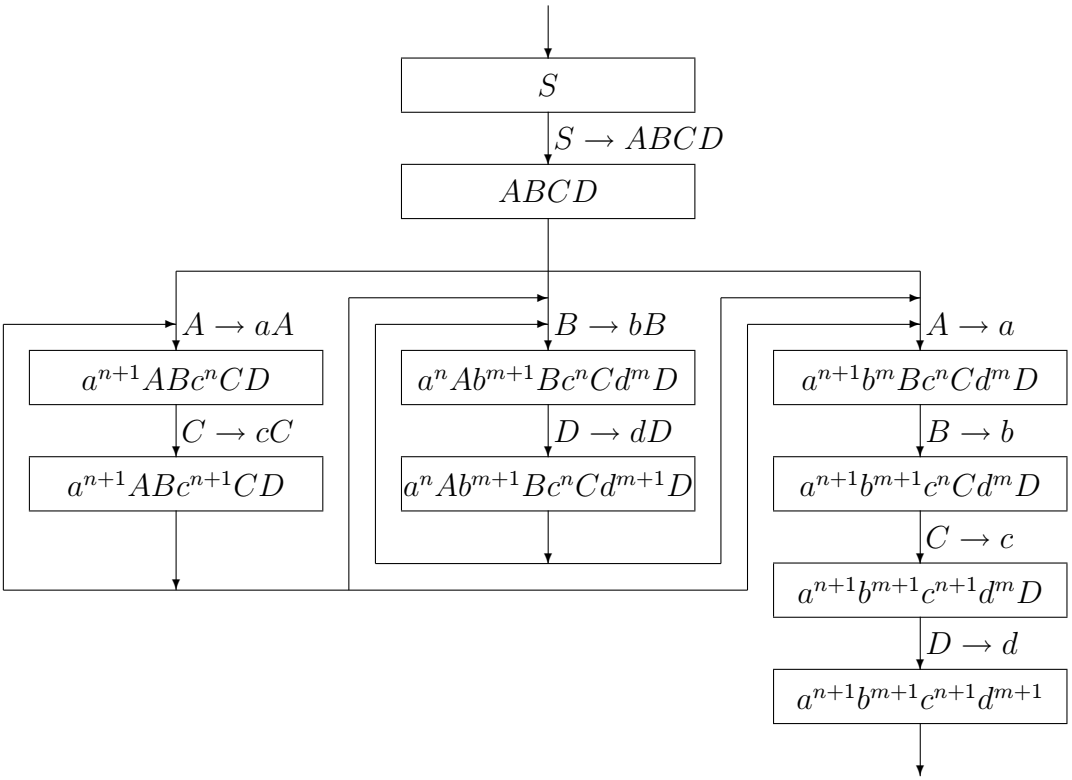
Příklad 2.5 $L(G) = \{a^m b^n a^m | 0 \leq n \leq m\}$

$G = (\{S, A, B, C\}, \{a, b\}, \{p_1, p_2, \ldots, p_7\}, S, R, \phi)$
$R = \{p_1(p_2p_4)^*(p_2p_3p_4)^*p_5p_6p_7\}$

$p_1 = S \to ABC$	$p_2 = A \to aA$	$p_3 = B \to bB$	$p_4 = C \to aC$
$p_5 = A \to \lambda$	$p_6 = B \to \lambda$	$p_7 = C \to \lambda$	

Po přepsání S na ABC se nejprve generuje $m - n$ dvojic terminálů a pomocí pravidel p_2 a p_4. Potom se n-krát opakují pravidla p_2, p_3, p_4. Tím se přidá zbytek terminálů a a všechny b. Nakonec jsou všechny neterminály odstraněny vypouštěcími pravidly.

Gramatiky pro jazyky $L = \{a^n b^n c^n | n \geq 1\}$ a $L = \{a^{2^m} | m \geq 1\}$ lze nalézt v [HFL–96].



Obrázek 2.5: Schéma průběhu derivace pro $a^nb^mc^nd^m,\ n, m \geq 1$ podle regulárně řízené gramatiky

2.1.3 Vektorová gramatika

Vektorová gramatika vypadá podobně jako maticová bez testování výskytu. Rozdíl je v užívání matic. Pokud se aplikuje některé pravidlo matice (nemusí být první), musí se aplikovat během derivace i ostatní pravidla. Není ale určeno, ve kterém kroku. Matice je tedy spíše nutné chápat jako množiny těch pravidel, která musí mít stejný počet aplikací v průběhu derivace.

Protože není přesně dáno pořadí užití pravidel, ztrácí význam testování výskytu. Vždy by se totiž našel krok v derivaci, kdy by se pravidlo v módu testování výskytu použít mohlo. Například, když již je slovo složeno jen z terminálních symbolů, mohla by se ve smyslu testování výskytu užít všechna pravidla. Pokud je potřeba nějaké pravidlo, které se někdy užije a někdy ne, již není splněna podmínka o stejném počtu užití pravidel v jedné matici. Takové pravidlo tak může být umístěno do vlastní matice.

Definice 2.4 *a) (Neuspořádaná) vektorová gramatika* (unordered vector grammar) *je čtveřice* $G = (N, T, M, S)$*, kde* N, T, M, S *jsou definovány stejně jako v maticových gramatikách.*

b) Jazyk $L(G)$ *generovaný neuspořádanou vektorovou gramatikou* G *je definován jako množina všech slov* $w \in T^*$ *takových, že existuje derivace*

$$S \Rightarrow_{p_1} w_1 \Rightarrow_{p_2} w_2 \Rightarrow_{p_3} \ldots \Rightarrow_{p_n} w,$$

kde $p_1 p_2 \dots p_n$ je permutace některého prvku množiny M^*.

λuV a uV označujeme rodiny neuspořádaných vektorových gramatik s vypouštějícími pravidly a bez nich.

Jestliže má vektorová gramatika všechny matice jednoprvkové, jedná se o bezkontextovou gramatiku, a proto $\mathcal{L}(CF) \subseteq \mathcal{L}(\lambda uV)$.

Maticové gramatiky v příkladech 2.1 a 2.2 jsou zároveň příklady vektorových gramatik. Rozdíl je jen v průběhu derivace. V maticových gramatikách bylo vždy na několik kroků určeno pořadí pravidel a potom se zvolilo pokračování. Tím byla zase nějaká předem daná sekvence několika pravidel. Ve vektorových gramatikách se v každém kroku může volit libovolné pravidlo. Podmínkou je jen, aby na konci derivace odpovídaly počty aplikací jednotlivých pravidel výskytům v maticích (aby všechna pravidla v jedné matici byla užita stejně často).

Problém nastává například u jazyka $L = \{ww | w \in \{a, b\}^+\}$. Není těžké vytvořit gramatiku generující dvě slova, která mají stejné terminály. Problém je zajistit, aby tyto terminály byly v obou slovech ve stejném pořadí. Vyplývá to ze zmiňované skutečnosti, že matice chápeme jako množiny pravidel, která musí mít stejný počet aplikací. Pořadí jejich užití, a tím i pořadí generování terminálů ve dvou slovech, není nijak určeno.

2.1.4 Programovaná gramatika

V programovaných gramatikách ovlivní užité pravidlo výběr pro další krok. K tomuto účelu má každé přiřazeny dvě podmnožiny množiny všech pravidel. Jedna podmnožina se nazývá pole úspěchu a druhá pole neúspěchu.

Pravidla programované gramatiky můžeme použít dvěma způsoby. Pokud má pravidlo neprázdné pole neúspěchu, lze jej aplikovat ve smyslu testování výskytu. Není-li levá strana pravidla obsažena ve slově, na které pravidlo používáme, zůstane toto slovo beze změny. V dalším kroku potom musíme užít některé z pravidel v množině φ (poli neúspěchu). Druhou možností je klasická aplikace pravidla, kdy je neterminál tvořící levou stranu ve slově nahrazen pravou stranou. Pro další krok derivace jsou povolena jen pravidla z množiny σ (pole úspěchu) přiřazené k právě užitému pravidlu. Při derivaci tak nemusíme hledat použitelné pravidlo mezi všemi, ale jen v podmnožině určené předchozím krokem.

Definice 2.5 *a) Programovaná gramatika* (programmed grammar) *je čtveřice $G = (N, T, P, S)$, kde*

- *N, T, S jsou definovány stejně jako v bezkontextových gramatikách a*
- *P je konečná množina trojic $r = (p, \sigma, \varphi)$, kde p je bezkontextové pravidlo, σ a φ jsou podmnožiny množiny P.*

b) Jazyk $L(G)$ generovaný programovanou gramatikou G je definován jako množina všech slov $w \in T^*$ takových, že existuje derivace

$$S = w_0 \Rightarrow_{r_1} w_1 \Rightarrow_{r_2} w_2 \Rightarrow_{r_3} \dots \Rightarrow_{r_k} w_k = w,$$

$k \geq 1$, *a pro* $r_i = (A_i \to v_i, \sigma_i, \varphi_i)$, $1 \leq i \leq k$, *platí jedna z následujících podmínek:*

$$w_{i-1} = w'_{i-1} A_i w''_{i-1},\ w_i = w'_{i-1} v_i w''_{i-1} \text{ pro nějaké } w'_{i-1}, w''_{i-1} \in V_G^*,\ r_{i+1} \in \sigma_i$$

nebo

$$A_i \text{ se nevyskytuje v } w_{i-1},\ w_{i-1} = w_i \text{ a } r_{i+1} \in \varphi_i.$$

c) Pokud pro všechna $r \in P$ *platí* $r = (p, \sigma, \phi)$*, říkáme, že* G *je programovaná gramatika bez testování výskytu. Jinak je* G *programovaná gramatika s testováním výskytu.*

Pro $r = (p, \sigma, \varphi)$ *nazýváme* σ *polem úspěchu a* φ *polem neúspěchu.*

λP_{ac} označujeme rodinu programovaných gramatik s testováním výskytu a vypouštějícími pravidly. Vynechání λ znamená zákaz vypouštějících pravidel, vynechání indexu ac znemožní testování výskytu.

Jestliže máme gramatiku bez testování výskytu a všechna pole úspěchu obsahují všechna pravidla, jde o bezkontextovou gramatiku. Třída programovaných gramatik je tedy nadmnožinou třídy bezkontextových gramatik.

Příklad 2.6 $L = \{a^n b^n c^n | n \geq 1\}$

$G = (\{S, A, B, C\}, \{a, b, c\}, \{f_1, \ldots, f_7\}, S)$
$f_1 = (S \to ABC, \{f_2, f_5\}, \phi)$ $f_2 = (A \to aA, \{f_3\}, \phi)$ $f_3 = (B \to bB, \{f_4\}, \phi)$
$f_4 = (C \to cC, \{f_2, f_5\}, \phi)$ $f_5 = (A \to a, \{f_6\}, \phi)$ $f_6 = (B \to b, \{f_7\}, \phi)$
$f_7 = (C \to c, \phi, \phi)$

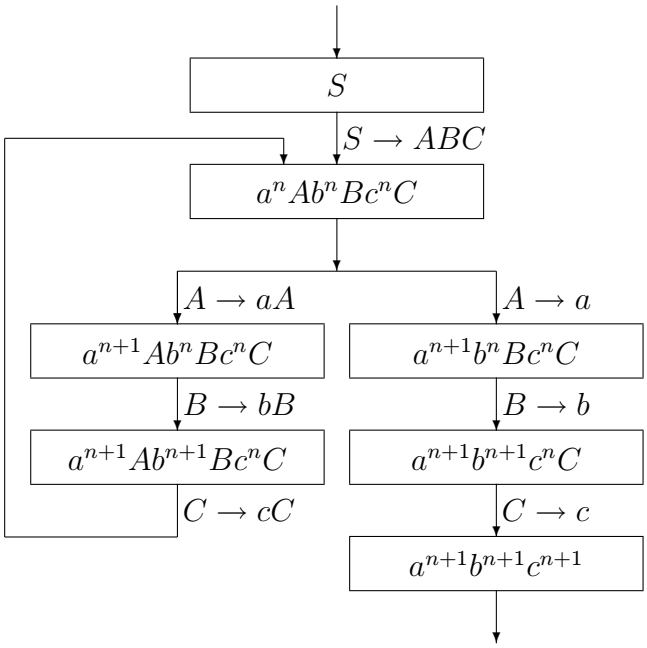
Na obrázku 2.6 je schéma průběhu derivace pro příklad 2.6. Po přepsání neterminálu S na ABC se v cyklu přidává jeden výskyt terminálu a, b i c. Volba nastává po použití pravidla, které má v příslušném poli více prvků. V našem příkladu to jsou f_4, které ukončuje jeden cyklus přidání terminálů, a počáteční f_1. Po jejich aplikaci je tří typů terminálů stejný počet a vybíráme, jestli budeme v derivaci pokračovat dále nebo ji ukončíme. V jiných případech je pořadí dáno jednoznačně, protože je v polích úspěchu ostatních pravidel jen jeden prvek.

Příklad 2.7 $L = \{a^n b^m c^n d^m | n, m \geq 1\}$

$G = (\{S, A, B, C, D\}, \{a, b, c, d\}, \{f_1, \ldots, f_9\}, S)$
$f_1 = (S \to ABCD, \{f_2, f_3, f_6\}, \phi)$ $f_2 = (A \to aA, \{f_4\}, \phi)$
$f_3 = (B \to bB, \{f_5\}, \phi)$ $f_4 = (C \to cC, \{f_2, f_3, f_6\}, \phi)$
$f_5 = (D \to dD, \{f_2, f_3, f_6\}, \phi)$ $f_6 = (A \to a, \{f_7\}, \phi)$
$f_7 = (B \to b, \{f_8\}, \phi)$ $f_8 = (C \to c, \{f_9\}, \phi)$
$f_9 = (D \to d, \phi, \phi)$

I když specifikace gramatiky v příkladu 2.7 je na první pohled odlišná od gramatiky v příkladu 2.1, obsahují stejná pravidla. Řízení zajišťuje navíc v obou případech stejný průběh derivace. Proto můžeme obrázek 2.1 považovat i za schéma průběhu derivace podle výše specifikované programované gramatiky.

Drobná úprava pravidla f_5 na $f_5 = (D \to dD, \{f_3, f_6\}, \phi)$ změní trochu průběh derivace. Gramatika generuje stejný jazyk a derivace probíhá jako v příkladu 2.4. Postup derivace byl znázorněn na obrázku 2.5.



Obrázek 2.6: Schéma průběhu derivace pro $a^n b^n c^n,\ n \geq 1$ podle programované gramatiky

Příklad 2.8 $L = \{ww | w \in \{a, b\}^+\}$

Tento příklad je vyřešen v [Sal–73] programovanou gramatikou bez testování výskytu. My si na něm předvedeme, jak lze testování výskytu využít.

$G = (\{S, A, B\}, \{a, b\}, \{f_1, \ldots, f_6\}, S)$

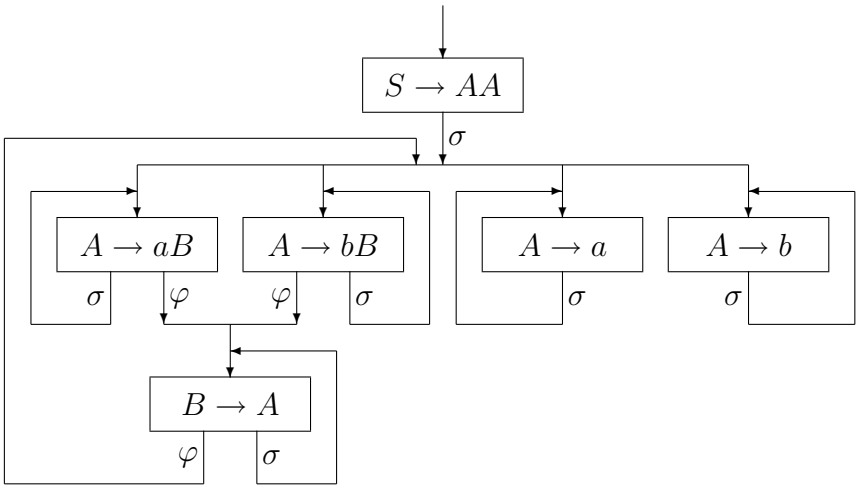
$f_1 = (S \to AA, \{f_2, f_3, f_5, f_6\}, \phi)$ $\quad f_2 = (A \to aB, \{f_2\}, \{f_4\})$

$f_3 = (A \to bB, \{f_3\}, \{f_4\})$ $\quad f_4 = (B \to A, \{f_4\}, \{f_2, f_3, f_5, f_6\})$

$f_5 = (A \to a, \{f_5\}, \phi)$ $\quad f_6 = (A \to b, \{f_6\}, \phi)$

Na obrázku 2.7 je zobrazeno, jaké pořadí pravidel při derivaci vynutí jednotlivá pole úspěchu a neúspěchu. Po přepsání počátečního symbolu na neterminály A můžeme volit ze čtyř možností pokračování derivace. Přepíšeme-li nějaké A, aplikuje se automaticky stejné pravidlo na všechny výskyty A. Pokud byla zvolena větev neukončující derivaci a všechna A jsou již odstraněna, aplikuje se vybrané pravidlo ve smyslu testování výskytu. Tím určíme pro další krok pravidlo přepisující B na A. I B se přepíší automaticky všechna. Když se ve větné formě žádné nevyskytuje, aplikuje se f_4 ve smyslu testování výskytu. V jeho poli neúspěchu jsou všechna pravidla s A na levé straně. Pro další krok tak můžeme opět vybrat libovolnou větev derivace. Konec nastane, když se zvolí přepis A na terminál a jsou takto odstraněna všechna A.

V uvedeném popisu se nikdy, kromě přepisu S, neříká, kolik symbolů A je ve větné formě, a kolik stejných slov tak bude generováno. To nám umožňuje využít pravidla f_2–f_6 jako součást nějaké rozsáhlejší gramatiky, ve které se symboly A vygenerují dynamicky v jiném počtu v různých derivacích. Příkladem je programovaná gramatika pro programovací jazyk, kterou si ukážeme v kapitole 4. Pro generování více výskytů identifikátorů tam použijeme stejný princip



Obrázek 2.7: Používání pravidel při derivaci $ww,\ w \in \{a,b\}^+$ podle programované gramatiky

jen s jinak pojmenovanými neterminály. Řešení v [Sal–73] vyžaduje pro každé opakování slova přidání jednoho neterminálu, čtyř pravidel a změnu polí úspěchu u stávajících pravidel. Využít takovou gramatiku na různý počet opakování slov je nemožné.

V [Sal–73] jsou uvedeny programované gramatiky pro jazyky $L = \{a^{2^m} | m \geq 1\}$ a $L = \{a^n b^m | 1 \leq n \leq m \leq 2^n\}$.

2.2 Řízení kontextovými podmínkami

V předchozí části jsme se zabývali gramatikami, ve kterých použité pravidlo ovlivnilo výběr pravidla pro další krok. Teď se zaměříme na jiný typ řízení. Výběr použitelného pravidla je založen na tvaru dosud vygenerované větné formy. Tedy možnost užití pravidla nezáleží jen na tom, jestli je levá strana obsažena ve větné formě, ale i na tom, jaké další symboly v něm jsou a nejsou obsaženy, v jakém jsou počtu a pořadí.

Při řízení pomocí sekvencí pravidel se často používala pravidla v módu testování výskytu. Protože by nemožnost užití jednoho pravidla v sekvenci zabránila užití celé sekvence, byla tento mód užitečný. Nyní se ale pravidla budou volit v každém kroku zvlášť a testování výskytu tak není nutné. Když se levá strana pravidla v derivované větné formě nevyskytuje, pravidlo se prostě nepoužije.

2.2.1 Podmíněná gramatika

První možností je volit použitelná pravidla podle celé aktuální větné formy. V praxi však není vhodné určit pro každou konkrétní větnou formu použitelná pravidla. Lepší je aplikaci povolit pro nějakou množinu slov. Konkrétně se používá regulární množina přiřazená ke každému pravidlu. V každém kroku derivace procházíme všechna pravidla a hledáme takové, které má levou stranu

obsaženou ve větné formě a celá větná forma patří do jeho množiny. Z pravidel splňujících tyto podmínky se vybere jedno a použije se.

Regulární množina se nejčastěji zadává regulárním výrazem nad množinou všech symbolů (terminálních i neterminálních) a větná forma musí vyhovovat tomuto regulárnímu výrazu, aby pravidlo mohlo být aplikováno.

Definice 2.6 *a) Podmíněná gramatika* (conditional grammar) *je čtveřice* $G = (N, T, P, S)$, *kde*

- N, T, S *jsou definovány stejně jako v bezkontextových gramatikách a*
- P *je konečná množina dvojic* $r = (p, R)$, *kde* p *je bezkontextové pravidlo a* R *je regulární množina nad* V_G.

b) Pro $x, y \in V_G^*$ *říkáme, že* x *přímo odvodí* y, *a značíme* $x \Rightarrow y$ *právě tehdy, když existuje dvojice* $r = (A \to w, R) \in P$ *taková, že* $x = x'Ax''$ *a* $y = x'wx''$ *pro nějaké* $x', x'' \in V_G^*$ *a* $x \in R$. *Někdy také značíme* $x \Rightarrow_r y$ *nebo* $x \Rightarrow_p y$, *když chceme určit dvojici nebo pravidlo, které je užito.*

c) Jazyk $L(G)$ *generovaný podmíněnou gramatikou* G *je definován jako*

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

Symbolem λC označujeme rodinu podmíněných gramatik. C je rodina podmíněných gramatik bez vypouštěcích pravidel.

Je zřejmé, že každá bezkontextová gramatika je snadno převeditelná na podmíněnou. Stačí ke každému pravidlu přiřadit množinu všech slov nad symboly terminální i neterminální abecedy. Potom je jasné, že větná forma se v této množině vyskytuje vždy a je možné používat pravidla stejně jako v bezkontextových gramatikách.

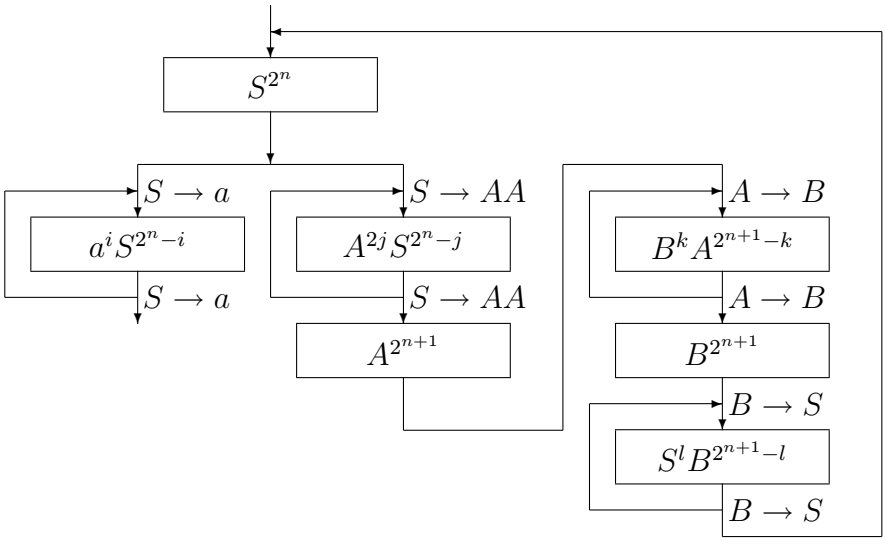
Příklad 2.9 $L = \{a^{2^n} | n \geq 0\}$

$G = (\{S, A, B\}, \{a\}, \{f_1, \ldots, f_4\}, S)$
$f_1 = (S \to AA, A^*S^+)$ $f_2 = (A \to B, B^*A^+)$
$f_3 = (B \to S, S^*B^+)$ $f_4 = (S \to a, a^*S^+)$

Na obrázku 2.8 je schéma průběhu derivace k příkladu 2.9. V cyklu se vždy zdvojnásobí počet neterminálů S. Nakonec se všechna S přepíší na terminál a.

Cyklus násobení neterminálů S začíná přepisováním každého S na AA. Regulární výrazy pravidel vynutí aplikaci pravidla f_1 postupně zleva. Kdyby nebyl přepsán vždy nejlevější neterminál S, derivace by se zablokovala. Žádné pravidlo by totiž nemělo splněnu podmínku danou regulárním výrazem. Dokud se neodstraní všechna S, nelze ani začít přepisovat A, protože v regulárním výrazu pravidla f_2 se S nevyskytuje.

Jsou-li přepsána všechna S, je neterminálů A dvojnásobek původního počtu S. Zdálo by se, že lze všechna A převést na S a začít další cyklus. Po přepsání několika A na S by však nebylo



Obrázek 2.8: Průběh derivace $a^{2^n},\ n \geq 0$ podle podmíněné gramatiky

jasné, ve které fázi derivace se nacházíme (jestli přepisujeme A na S nebo S na A). Podařilo by se tak vygenerovat slovo s libovolným počtem a.

Zmiňovanému problému se vyhneme, když nejprve všechna A přepíšeme na B a až potom B na S. Takto je to v našem řešení příkladu i na obrázku. Přepis A na B i B na S se provádí postupně zleva, což opět zajišťuje tvar regulárních výrazů u pravidel. Po získání větné formy složené pouze z neterminálů S začne cyklus znovu nebo se tato S přepíší na a a derivace končí.

Příklad 2.10 $L = \{ww | w \in \{a,b\}^+\}$

$G = (\{S, A, X, V, W, Z\}, \{a, b\}, \{f_1, \ldots, f_9\}, S)$

$f_1 = (S \to AA, S)$
$f_2 = (Z \to A, (a+b)^*(Z+A)(a+b)^*Z)$
$f_3 = (A \to aX, (a+b)^*((aX)+A)(a+b)^*A)$
$f_4 = (A \to V, (a+b)^*(A+V)(a+b)^*A)$
$f_5 = (A \to bX, (a+b)^*((bX)+A)(a+b)^*A)$
$f_6 = (A \to W, (a+b)^*(A+W)(a+b)^*A)$
$f_7 = (X \to Z, (a+b)^*(X+Z)(a+b)^*X)$
$f_8 = (V \to a, (a+b)^*V^*(a+b)^*V)$
$f_9 = (W \to b, (a+b)^*W^*(a+b)^*W)$

Tvorba dvojice stejných slov v příkladu 2.10 je založena na uplatňování stejných pravidel na dvojici neterminálů. Na začátku derivace tuto dvojici vytvoříme pravidlem f_1.

Jsou-li ve větné formě dvě A, můžeme si vybrat čtyři možnosti pokračování. Jsou jimi pravidla f_3–f_6. Zvolíme-li kterékoli z nich, musíme ho aplikovat nejprve na levý výskyt A a potom na pravý. Regulární výrazy nám jinou možnost nedávají. Po přepsání pravého výskytu by tvar větné formy neumožnil aplikaci žádného pravidla. Pokud bylo zvoleno f_4 nebo f_6, zavedly se neterminály V nebo W. Ty je možné převést jen na terminály a derivace končí.

Bylo-li zvoleno pravidlo f_3 nebo f_5, objevila se ve větné formě dvojice neterminálů X. Ty nemůžeme přepsat přímo na A a začít další cyklus. Kdyby existovalo pravidlo $X \to A$ a použilo

se na pravý výskyt X, vznikla by větná forma vyhovující podmínce pravidla f_3 nebo f_5. Po jeho užití by druhé slovo bylo o jeden terminál delší než první. Této situaci se v řešení příkladu vyhýbáme přepisem X na Z a až potom Z na A. Pravidla f_7 a f_2 se vždy použijí dvakrát za sebou. Napřed se přepíše levý výskyt neterminálu a potom pravý. Opět to zajistí tvar množin u pravidel. Až jsou ve větné formě dvě A, volí se znovu ze čtyř možností.

2.2.2 Polopodmíněná gramatika

Polopodmíněná gramatika rozhoduje, stejně jako podmíněná, o aplikovatelných pravidlech na základě větné formy. Nekontroluje ji ale jako celek. Pouze hledá výskyt určitých řetězců terminálů a neterminálů v ní. Použití pravidla je podmíněno právě existencí nějakých podslov derivované větné formy. Jiná podslova aplikaci pravidla naopak zabraňují.

Pro spojení s podslovy přiřazuje polopodmíněná gramatika každému pravidlu dvě množiny. Prvky těchto množin mohou být libovolná slova nad symboly abecedy (terminálními i neterminálními). Jedna z množin se nazývá povolený kontext. Každé slovo z ní se musí vyskytovat ve větné formě, na kterou chceme pravidlo aplikovat. Jedná se tedy spíše o vynucený, než povolený kontext. Druhá množina má opačný význam. Žádné slovo z ní se nesmí ve větné formě vyskytovat, abychom pravidlo mohli užít.

Při derivaci hledáme v každém kroku použitelná pravidla. Jejich levá strana se musí vyskytovat ve větné formě. Pro každé takové musíme projít jeho povolený kontext a vyhledat slova v něm obsažená ve větné formě. Pokud se to podaří, musíme ještě prokázat, že se ve větné formě nevyskytuje žádné slovo ze zakázaného kontextu. Až po splnění všech tří podmínek je pravidlo aplikovatelné. Je-li takových více, můžeme rozhodnout o dalším průběhu derivace.

Definice 2.7 *a) Polopodmíněná gramatika* (semi-conditional grammar) *je čtveřice* $G = (N, T, P, S)$*, kde*

- N, T, S *jsou definovány stejně jako v bezkontextových gramatikách*
- P *je konečná množina trojic* $r = (p, R, Q)$*, kde* p *je bezkontextové pravidlo,* R *a* Q *jsou disjunktní konečné množiny slov nad* V_G*.*

b) Pro $x, y \in V_G^*$ *říkáme, že* x *přímo odvodí* y*, a značíme* $x \Rightarrow y$*, právě tehdy, když existuje trojice* $r = (A \to w, R, Q) \in P$ *taková, že* $x = x'Ax''$ *a* $y = x'wx''$ *pro nějaké* $x', x'' \in V_G^*$*, každé slovo z množiny* R *je podslovem* x *a žádné slovo z množiny* Q *není podslovem* x*. Někdy také značíme* $x \Rightarrow_r y$ *nebo* $x \Rightarrow_p y$*, když chceme určit trojici nebo pravidlo, které je užito.*

c) Jazyk $L(G)$ *generovaný polopodmíněnou gramatikou* G *je definován jako*

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

Symbolem λsC označujeme rodinu polopodmíněných gramatik. sC je rodina polopodmíněných gramatik bez vypouštěcích pravidel.

I polopodmíněné gramatiky zahrnují bezkontextové. Ke každé bezkontextové gramatice se jednoduše vytvoří polopodmíněná přiřazením dvou prázdných množin ke každému pravidlu. Před použitím pravidla se tak nic nekontroluje a chování gramatiky zůstává stejné, jaké má bezkontextová.

Příklad 2.11 $L = \{a^{2^n} | n \geq 0\}$

$G = (\{S, A, B\}, \{a\}, \{f_1, \ldots, f_4\}, S)$
$f_1 = (S \to AA, \phi, \{B, a, SA\})$ $f_2 = (A \to B, \phi, \{AB, S\})$
$f_3 = (B \to S, \phi, \{A, BS\})$ $f_4 = (S \to a, \phi, \{A, B, Sa\})$

Derivace podle gramatiky v příkladu 2.11 probíhá i přes jiný typ řízení stejně jako v příkladu 2.9. Dvojice AB, BS a Sa, vyskytující se v zakázaných kontextech pravidel bychom mohli vypustit. Generovaný jazyk by se nezměnil. Během derivace by ale nebyl zajištěn přepis levého výskytu neterminálu před pravým výskytem stejného neterminálu. Když tyto dvojice v množinách jsou a přepíše se pravý výskyt, není možné stejné pravidlo na levý výskyt použít. Protože ostatní pravidla nelze aplikovat z jiných důvodů, je derivace zablokovaná.

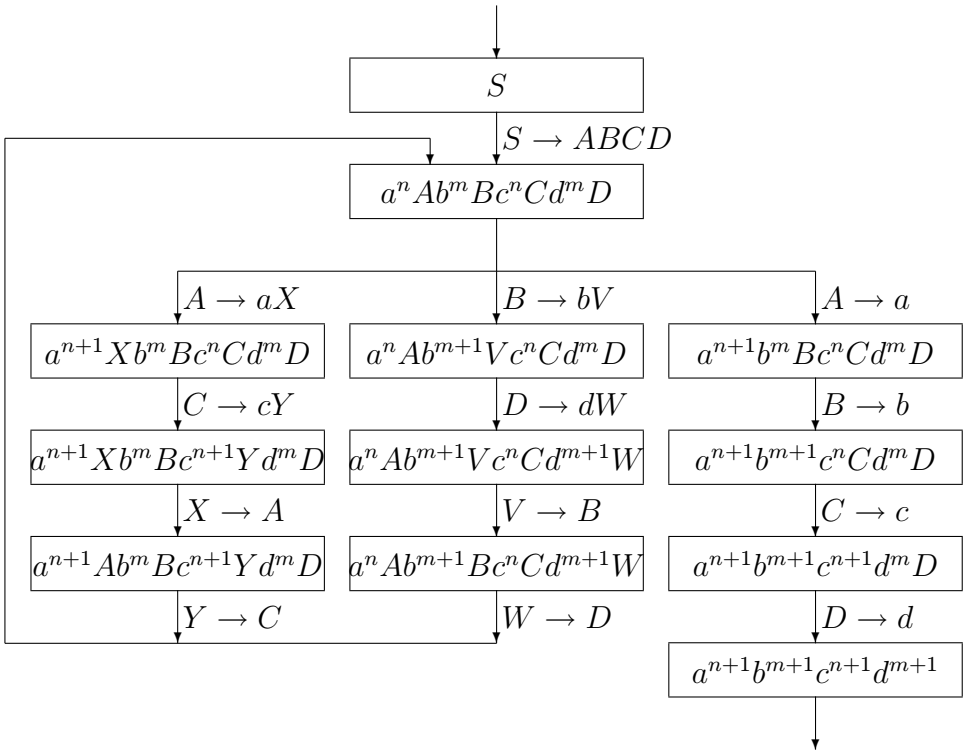
Příklad 2.12 $L(G) = \{a^n b^m c^n d^m | n, m \geq 1\}$

$G = (\{S, A, B, C, D, X, Y, V, W\}, \{a, b, c, d\}, \{p_1, p_2, \ldots, p_{13}\}, S)$
$p_1 = (S \to ABCD, \phi, \phi)$ $p_2 = (A \to aX, \{B, C, D\}, \phi)$
$p_3 = (B \to bV, \{A, C, D\}, \phi)$ $p_4 = (C \to cY, \{X\}, \phi)$
$p_5 = (D \to dW, \{V\}, \phi)$ $p_6 = (A \to a, \{B, C, D\}, \phi)$
$p_7 = (B \to b, \{C, D\}, \{X, A\})$ $p_8 = (C \to c, \phi, \{A, B, X\})$
$p_9 = (D \to d, \phi, \{B, V, C\})$ $p_{10} = (X \to A, \{Y\}, \phi)$
$p_{11} = (Y \to C, \{A\}, \phi)$ $p_{12} = (V \to B, \{W\}, \phi)$
$p_{13} = (W \to D, \{B\}, \phi)$

Na obrázku 2.9 je zobrazen průběh derivace podle gramatiky uvedené v příkladu 2.12. Srovnáme-li řešení s příkladem 2.1, zjistíme, že základ je podobný. V maticové gramatice jsme však měli „historii“ použitých pravidel danou maticemi. V polopodmíněné gramatice při použití pravidla nevíme, které bylo aplikováno v předchozím kroku. Proto pravidla p_2–p_5 zavádějí nové neterminály. Podle výskytu těchto neterminálů ve větné formě poznáme, která pravidla byla aplikována v předchozích krocích. Například, když je ve větné formě X a není Y, bylo použito p_2 a musíme aplikovat p_4, aby a a c přibývaly stejně. Podobné je to s pravidly p_3 a p_5.

Přidané neterminály musíme zpátky převést na původní, aby mohla derivace pokračovat cyklicky stejným postupem dále. K tomuto převedení slouží pravidla p_{10}–p_{13}.

Některé neterminály z povolených i zakázaných kontextů pravidel bychom mohli odstranit a generovaný jazyk by se nezměnil. Derivace by potom probíhala méně deterministicky. Například by se mohlo přidat a, potom několik b, d a až dodatečně c. Pro automatické provádění derivace je méně nedeterminismů vhodnější, protože není potřeba tak často nechávat uživatele volit pokračování.



Obrázek 2.9: Schéma průběhu derivace pro $a^n b^m c^n d^m$, $n, m \geq 1$ podle polopodmíněné gramatiky

V povoleném kontextu p_{10} se nachází Y, aby se zajistilo, že před přepisem X již byl vygenerován terminál c a s ním neterminál Y. A v povoleném kontextu p_{11} zase zajistí, aby nejprve bylo použito p_{10} a až po něm p_{11}. Kdybychom to nezajistili, byla by možná následující derivace:

$$a^{n+1} X b^m B c^{n+1} Y d^m D \Rightarrow_{p_{11}} a^{n+1} X b^m B c^{n+1} C d^m D \Rightarrow_{p_4} a^{n+1} X b^m B c^{n+2} Y d^m D$$

Počty a a c by se potom mohly lišit až do konce derivace a gramatika by negenerovala požadovaný jazyk. Stejný princip je použit i pro dvojici b, d.

2.2.3 Gramatika s náhodným kontextem

Gramatiky s náhodným kontextem (random context grammars) jsou speciálním příkladem polopodmíněných gramatik. Povolený i zakázaný kontext tvoří podmnožiny množiny neterminálních symbolů. Nevyskytují se tam tedy terminály. Neterminální symboly můžeme chápat jako slova délky jedna a všechny gramatiky s náhodným kontextem tak splňují definici polopodmíněné gramatiky.

Symbolem λRC_{ac} označujeme rodinu gramatik s náhodným kontextem. Pokud uvažujeme rodinu gramatik neobsahujících vypouštěcí pravidla, vynecháme symbol λ. Uvažujeme-li rodinu gramatik, kde všechny zakázané kontexty jsou prázdné, vynecháme index ac.

Příklad 2.12 na polopodmíněnou gramatiku je současně i příkladem gramatiky s náhodným kontextem. Všechny množiny tvořící povolený i zakázaný kontext totiž obsahují jen neterminály. Pro příklad 2.11 to již neplatí, protože se v zakázaných kontextech nachází dvojice symbolů. Slouží ale pouze k zajištění přepsání nejlevějšího výskytu daného symbolu a daly by se odstranit bez změny generovaného jazyka. V pravidle f_1 je v zakázaném kontextu terminál. Zajišťuje, že nemůžeme množit symboly S, pokud byl již některý přepsán na terminál. Odporuje však definici gramatiky s náhodným kontextem.

Problém vyřešíme zavedením dalšího neterminálu X. Všechna S se při ukončování derivace převedou na něj a až se ve větné formě žádné S nevyskytuje, přepíšeme všechna X na a. Do zakázaných kontextů se tak dostanou jen neterminály. Celé řešení převzaté z [HFL–96] je uvedeno v příkladu 2.13.

Příklad 2.13 $L = \{a^{2^n} | n \geq 0\}$

$G = (\{S, A, B, X\}, \{a\}, \{f_1, \ldots, f_5\}, S)$
$f_1 = (S \to AA, \phi, \{B, X\})$ $f_2 = (A \to B, \phi, \{S\})$ $f_3 = (B \to S, \phi, \{A\})$
$f_4 = (S \to X, \phi, \{A\})$ $f_5 = (X \to a, \phi, \{S\})$

2.2.4 Uspořádaná gramatika

Nabízí se možnost přiřazení priorit jednotlivým pravidlům. V každém kroku derivace je potom vybráno a aplikováno z použitelných to nejprioritnější. Tento mechanismus využívají uspořádané gramatiky.

Formálně se zavádí relace uspořádání nad pravidly. Pravidlo se nesmí aplikovat, je-li možno použít jiné, větší podle uspořádání. V každém kroku derivace se najdou aplikovatelná pravidla klasickým způsobem (levá strana se musí nacházet ve větné formě). Z takto vybraných se určí maximální pravidlo nebo pravidla. Jedno z maximálních potom použijeme.

Uspořádání může být neostré (dvě pravidla mohou mít stejnou prioritu) a nemusí být úplné (některé pravidlo není porovnatelné s jiným). Tím se zajistí možnost volby při derivaci. Kdyby bylo úplné ostré, muselo by se vždy použít největší aplikovatelné pravidlo a bylo by možné vygenerovat gramatikou nejvýše jedno slovo (derivace by byla deterministicky určena uspořádáním).

Definice 2.8 *a) Uspořádaná gramatika* (ordered grammar) *je čtveřice* $G = (N, T, P, S)$*, kde*

- N, T, S *jsou definovány stejně jako v bezkontextových gramatikách a*
- P *je konečná (částečně) uspořádaná množina bezkontextových pravidel.*

b) Pro $x, y \in V_G^*$ *říkáme, že* x *přímo odvodí* y*, a značíme* $x \Rightarrow y$ *právě tehdy, když existuje pravidlo* $p = A \to w \in P$ *takové, že* $x = x'Ax''$*,* $y = x'wx''$*, a neexistuje pravidlo* $q = B \to v \in P$ *takové, že* $p < q$ *a* B *se vyskytuje v* x*. Pokud chceme určit použité pravidlo, píšeme* $x \Rightarrow_p y$

c) Jazyk $L(G)$ *generovaný uspořádanou gramatikou* G *je definován jako*

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

Rodinu uspořádaných gramatik s vypouštěcími pravidly, resp. bez nich, označujeme λO, resp. O.

Opět není těžké si uvědomit, že třída uspořádaných gramatik obsahuje třídu bezkontextových. Každá bezkontextová gramatika je vlastně uspořádanou gramatikou se všemi pravidly se stejnou prioritou, popř. se všemi pravidly neporovnatelnými.

Příklad 2.14 $L = \{ww | w \in \{a,b\}^+\}$

$G = (\{S,A,B,C,V,X,Y,Z\}, \{a,b\}, \{f_1,\ldots,f_{16}\}, S)$

$f_1 = S \to AA \quad f_2 = A \to aX \quad f_3 = A \to bY \quad f_4 = A \to B$
$f_5 = A \to C \quad f_6 = V \to A \quad f_7 = Y \to V \quad f_8 = X \to V$
$f_9 = B \to a \quad f_{10} = C \to b \quad f_{11} = A \to Z \quad f_{12} = X \to Z$
$f_{13} = Y \to Z \quad f_{14} = B \to Z \quad f_{15} = C \to Z \quad f_{16} = V \to Z$

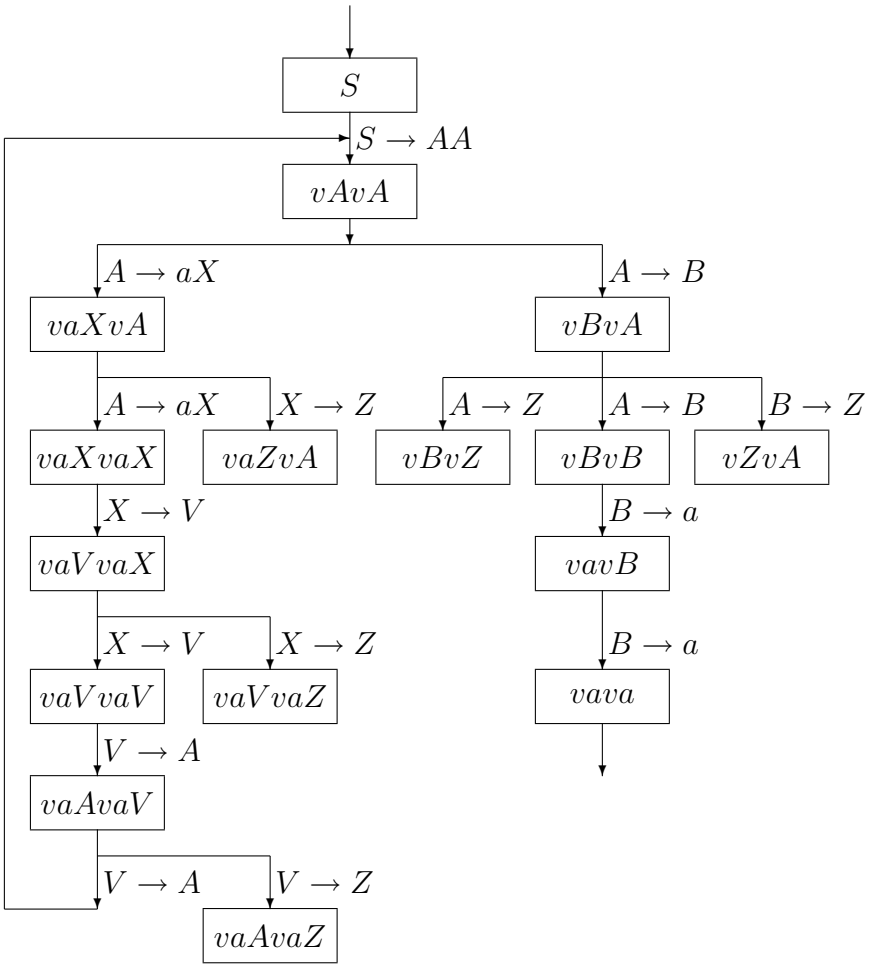
$f_{11} > f_7, f_8, f_9, f_{10} \quad f_{15} > f_2, f_3, f_4 \quad f_{14} > f_2, f_3, f_5$
$f_{16} > f_2, f_3, f_4, f_5 \quad f_{13} > f_2, f_4, f_5, f_6 \quad f_{12} > f_3, f_4, f_5, f_6$

Na obrázku 2.10 je zobrazena část schématu derivace podle gramatiky uvedené v příkladu 2.14. Nejsou zde větve vedoucí k přidání terminálů b. Jsou však téměř stejné jako větve pro přidání a, jen označení neterminálů je odlišné. Také je v zobrazení vždy přepsán dříve levý výskyt neterminálu před pravým. Můžeme si však všimnout, že pořadí, v jakém jsou pravidla aplikována na jednotlivé výskyty, nehraje ve výsledku derivace žádnou roli.

Pro zajištění potřebné derivace je použit jednoduchý trik. Pravidlům, která si nepřejeme v daném kroku použít, dáme prioritu menší než nějakému pomocnému pravidlu. Toto pomocné pravidlo přepíše některý neterminál na jiný, který se nevyskytuje na levé straně žádného pravidla. Derivace potom nemůže vést k terminálnímu slovu a je zablokovaná. Na obrázku tomu odpovídají větné formy, ze kterých nevede žádná šipka. Pomocí tohoto obratu jsme docílili, že vždy je aplikovatelné pouze jedno pravidlo neblokující derivaci. Výjimkou je pouze situace, kdy jsou ve větné formě dvě A. Zde se rozhoduje o pokračování derivace. Volbou jsou pravidla f_4 a f_5 směřující k ukončení derivace, nebo pravidla f_2 a f_3 vedoucí k vygenerování další dvojice terminálů.

Na první pohled nemusí být zřejmé, proč jsou přidána prioritnější pravidla přepisující nějaký neterminál na pomocné Z. Přepsání druhého neterminálu stejným pravidlem jako prvního by se mohlo zajistit uspořádáním přímo mezi pravidly, která na dvojici můžeme aplikovat. Nastane však následující problém (terminály v pravidlech v příkladu nyní nejsou podstatné):

- $AA \Rightarrow XA \Rightarrow XX$ si vynutí $A \to X > X \to V$, aby se nepřepisovalo X, dokud není přepsáno druhé A
- $XX \Rightarrow VX \Rightarrow VV$ si vynutí $X \to V > V \to A$, aby se nepřepisovalo V, dokud není přepsáno druhé X
- $VV \Rightarrow AV \Rightarrow AA$ si vynutí $V \to A > A \to X$, aby se nepřepisovalo A, dokud není přepsáno druhé V



Obrázek 2.10: Průběh derivace $ww,\ w \in \{a,b\}^+$ podle uspořádané gramatiky

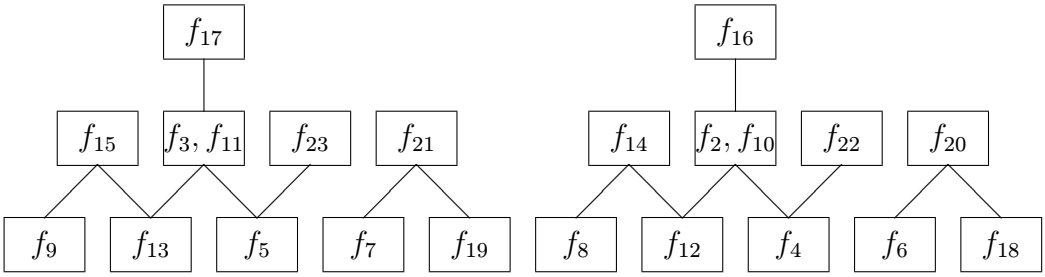
Uvědomíme-li si tranzitivnost relace uspořádání, zjistíme, že každé pravidlo z uvedené trojice je větší (a zároveň menší) než ono samo. Tím vzniká spor, kterému se vyhýbáme zmíněnými pomocnými pravidly blokujícími derivaci.

Uspořádání mezi pravidly můžeme někdy přehledněji zobrazit pomocí Haaseova diagramu. Ukážeme si to na následujícím příkladu.

Příklad 2.15 $L = \{a^n b^m c^n d^m | m, n \geq 1\}$

$G = (\{S, A, B, C, D, T, U, V, W, X, Y, Z\}, \{a, b, c, d\}, \{f_1, \ldots, f_{23}\}, S)$

$f_1 = S \to ABCD$	$f_2 = A \to aX$	$f_3 = B \to bV$	$f_4 = C \to cY$	$f_5 = D \to dW$
$f_6 = X \to A$	$f_7 = V \to B$	$f_8 = Y \to C$	$f_9 = W \to D$	$f_{10} = A \to U$
$f_{11} = B \to T$	$f_{12} = C \to c$	$f_{13} = D \to d$	$f_{14} = X \to Z$	$f_{15} = V \to Z$
$f_{16} = Y \to Z$	$f_{17} = W \to Z$	$f_{18} = U \to a$	$f_{19} = T \to b$	$f_{20} = C \to Z$
$f_{21} = D \to Z$	$f_{22} = U \to Z$	$f_{23} = T \to Z$		



Gramatika příkladu 2.15 využívá pro zajištění potřebného pořadí pravidel při derivaci podobný mechanismus jako příklad 2.14. Jsou zde opět pravidla přepisující některé neterminály na Z, které je neodstranitelné a blokuje derivaci. Čtyři základní větve derivace vypadají následovně:

a) $a^n Ab^m Bc^n Cd^m D \Rightarrow_{f_2} a^{n+1} Xb^m Bc^n Cd^m D \Rightarrow_{f_4} a^{n+1} Xb^m Bc^{n+1} Yd^m D \Rightarrow_{f_6}$
$a^{n+1} Ab^m Bc^{n+1} Yd^m D \Rightarrow_{f_8} a^{n+1} Ab^m Bc^{n+1} Cd^m D$

b) $a^n Ab^m Bc^n Cd^m D \Rightarrow_{f_3} a^n Ab^{m+1} Vc^n Cd^m D \Rightarrow_{f_5} a^n Ab^{m+1} Vc^n Cd^{m+1} W \Rightarrow_{f_7}$
$a^n Ab^{m+1} Bc^n Yd^{m+1} W \Rightarrow_{f_9} a^n Ab^{m+1} Bc^n Cd^{m+1} D$

c) $a^n Ab^m Bc^n Cd^m D \Rightarrow_{f_{10}} a^n Ub^m Bc^n Cd^m D \Rightarrow_{f_{12}} a^n Ub^m Bc^{n+1} d^m D \Rightarrow_{f_{18}}$
$a^{n+1} b^m Bc^{n+1} d^m D$

d) $a^n Ab^m Bc^n Cd^m D \Rightarrow_{f_{11}} a^n Ab^m Tc^n Cd^m D \Rightarrow_{f_{13}} a^n Ab^m Tc^n Cd^{m+1} \Rightarrow_{f_{19}}$
$a^n Ab^{m+1} c^n Cd^{m+1}$

Větev a) se může opakovat libovolně, dokud není použita větev c). Podobně b) lze opakovat do použití d). Uspořádání pravidel nebrání, aby se posloupnost b) nebo d) prolínala s větví a) nebo c). Tedy ne vždy musí derivace probíhat přesně podle popsaných posloupností. Pořadí pravidel v nich zůstane vždy zachováno a větev se uplatní celá, ale mezi kroky mohou být vloženy aplikace pravidel podle jiné větve.

Promysleme podrobněji, proč posloupnost a) proběhne zrovna tak, jak je to popsáno. Neterminály B a D se zabývat nebudeme, protože jsou využity jen v jiných větvích. Na začátku máme A a C. C nemůžeme přepsat, protože f_{12} i f_4 jsou menší než f_2. Na A můžeme aplikovat f_2 a f_{10}. Zvolili jsme f_2. Tím získáme X a C. Aplikace f_6 není možná, protože f_{20} blokující derivaci je větší. Musíme tedy přepsat C. K dispozici jsou f_4 a f_{12}. f_{12} je ale menší než f_{14}, které lze aplikovat a blokuje derivaci. To vynutí užití f_4. Z dvojice X, Y musí být nejprve pravidlo aplikováno na X, protože $f_8 < f_{14}$. Na X tedy užijeme f_6. V situaci A a Y je zajištěno přepsání Y na C tím, že oba pravidla použitelná na A jsou menší než f_{16}. Tím větev končí a může proběhnout znovu. Stejný princip s jinak pojmenovanými pravidly a neterminály je použit ve větvi b), podobný i v c) a d).

Další příklady uspořádaných gramatik jsou uvedeny v [Sal–73] ($L = \{a^n b^n c^n | n \geq 1\}$) a [HFL–96] ($L = \{a^{2^m} | m \geq 1\}$).

2.3 Gramatiky s částečným paralelismem

Všechny gramatiky, kterými jsme se dosud zabývali, aplikovaly pravidla sekvenčně. V každém kroku se přepsal nejvýše jeden neterminál. Jen při použití pravidla v módu testování výskytu se nepřepsalo nic.

V této části budeme uvažovat možnost, kdy se přepisuje několik neterminálů současně v jednom kroku derivace. Typů gramatik je opět několik. Liší se tím, jestli nutí používat v jednom kroku stejná pravidla nebo různá, jestli jsou pravidla v jednom kroku aplikována na stejný neterminál nebo různé apod.

2.3.1 Indická paralelní gramatika

Indická paralelní gramatika se zadává stejně jako bezkontextová. Rozdíl je jen v průběhu derivace. I výběr použitelných pravidel probíhá stejně jako v bezkontextové gramatice. Pouze se zjistí, jestli se levá strana vyskytuje ve větné formě. Nemusí být splněny žádné další podmínky, které jsme uvažovali u všech dosud zmíněných gramatik. V bezkontextové gramatice se vybrané pravidlo aplikuje na jeden z výskytů levé strany. Indická paralelní gramatika vybrané pravidlo použije v jednom kroku současně na všechny výskyty.

Definice 2.9 *a) Indická paralelní gramatika* (Indian paralel grammar) *je čtveřice* $G = (N, T, P, S)$*, kde* N, T, P, S *jsou definovány stejně jako v bezkontextových gramatikách.*

b) Pro $x, y \in V_G^*$ *říkáme, že* x *přímo odvodí* y*, a značíme* $x \Rightarrow y$ *právě tehdy, když platí následující podmínky:*

- $x = x_1 A x_2 A \dots x_n A x_{n+1}, \quad n \geq 1,\ A \in N,\ x_i \in (V_G - \{A\})^*$*, pro* $1 \leq i \leq n+1$*,*
- $y = x_1 w x_2 w \dots x_n w x_{n+1}$*,*
- $A \to w \in P$

Pokud chceme specifikovat použité pravidlo $p = A \to w$*, užíváme značení* $x \Rightarrow_p y$*.*

c) Jazyk $L(G)$ *generovaný indickou paralelní gramatikou* G *je definován jako*

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

Rodinu Indických paralelních gramatik označujeme λIP, resp. IP neuvažujeme-li vypouštějící pravidla.

Je dokázáno, že třídy bezkontextových jazyků a jazyků generovaných indickými gramatikami jsou neporovnatelné. Existují kontextové jazyky, které lze generovat indickou paralelní gramatikou (např. příklady 2.16 a 2.17), ale také bezkontextové jazyky, které indickou gramatikou generovatelné nejsou (např. tzv. Dyckův jazyk - viz. [Sky–74]).

Příklad 2.16 $L = \{a^{2^m} | m \geq 0\}$

$G = (\{S, A\}, \{a\}, \{A \to AA, A \to a\}, A)$

Příklad 2.16 ukazuje výhodu aplikace stejného pravidla na všechny výskyty neterminálu. Vystačíme si jen se dvěma pravidly. Jedno aplikované na všechna A jejich počet zdvojnásobí. Druhé pravidlo potom všechny neterminály v jednom kroku převede na terminály a a derivace končí. Slovo a^{2^m} je tak vygenerováno během $m + 1$ kroků (po m krocích získáme větnou formu A^{2^m} a další krok derivaci ukončí přepisem na terminály).

Protože řízení je v indických gramatikách založeno pouze na aplikaci stejného pravidla na všechny výskyty levé strany ve větné formě, nedokáží generovat jazyky typu

$$L_1 = \{a^n b^n c^n | n \geq 1\} \text{ nebo } L_2 = \{a^n b^m c^n d^m | m, n \geq 1\}.$$

V nich potřebujeme generovat stejný počet různých terminálů. Tyto různé terminály nemůžeme generovat stejným pravidlem a vztah mezi užitím různých pravidel nám gramatika neposkytuje. V příkladu 2.17 je uveden jazyk podobný na L_2, který ale hlídá počet stejných terminálů. To nám indická gramatika umožňuje. V jednom kroku vždy přepíše oba neterminály A nebo B podle stejného pravidla.

Příklad 2.17 $L = \{a^n b^m a^n b^m | n, m \geq 1\}$

$G = (\{S, A, B\}, \{a, b\}, \{S \to ABAB, A \to aA, B \to bB, A \to a, B \to b\}, S)$

2.3.2 k-gramatika

k-gramatika je zadána opět stejně jako bezkontextová. I použitelná pravidla v každém kroku se určují stejně. Navíc je však součástí specifikace konkrétní k-gramatiky přirozené číslo k. To určuje, kolik neterminálů je nutné přepsat současně v každém kroku. Na rozdíl od indických paralelních gramatik mohou být pravidla použitá v jednom kroku různá. I neterminály, na které jsou aplikována, se mohou lišit. Pokud nastane situace, kdy je ve větné formě neterminálů méně než k, je derivace zablokovaná. Výjimku tvoří první krok. Protože počáteční neterminál je jen jeden, může se aplikovat najednou právě jedno pravidlo. To však musí počáteční neterminál nahradit větnou formou s alespoň k neterminály.

Definice 2.10 *a) k-gramatika* (k-grammar) *je čtveřice $G = (N, T, P, S)$, kde N, T, P, S jsou definovány stejně jako v bezkontextových gramatikách.*

b) Pro $x, y \in V_G^*$ říkáme, že x přímo odvodí y, a značíme $x \Rightarrow y$ právě tehdy, když

$$x = S \text{ a } S \to y \in P$$

nebo

$$x = x_1 A_1 x_2 A_2 \ldots x_k A_k x_{k+1}, \text{ pro nějaká } x_1, x_2, \ldots, x_{k+1} \in V_G^*,$$
$$y = x_1 w_1 x_2 x_2 \ldots x_k w_k x_{k+1},$$
$$A_1 \to w_1,\ A_2 \to w_2, \ldots, A_k \to w_k \in P.$$

c) Jazyk $L(G)$ generovaný k-gramatikou G je definován jako

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

Rodinu k-gramatik označujeme λkG, resp. kG neuvažujeme-li vypouštějící pravidla.

Příklad 2.18 $L = \{a^m b^n a^m | 0 \leq n \leq m\}$

$G = (\{S, A, B\}, \{a, b\}, \{S \to ABA, A \to aA, B \to bB, B \to B, A \to \lambda, B \to \lambda\}, S)$ pro $k = 3$

Podle gramatiky v příkladu 2.18 se v prvním kroku derivace zavedou tři neterminály. Ty jsou potom díky $k = 3$ přepisovány v každém kroku všechny. Aby se derivace nezablokovala, musí po každém kroku zůstat větná forma se třemi neterminály (více jich pomocí pravidel této konkrétní gramatiky získat nejde) nebo terminální slovo. Kromě posledního kroku se vždy vytvoří dvojice a (neterminál A lze přepsat jen podle $A \to aA$). Symbol B se může přepsat podle dvou pravidel. Počet terminálů b se tak v některých krocích zvětšit nemusí a na konci jich je méně nebo stejně jako a. V posledním kroku derivace se všechny neterminály současně přepíší na λ. Kdyby se neodstranily vypouštěcím pravidlem všechny najednou, nebylo by možné pokračovat v derivaci.

k-gramatika zajistí, že v jednom kroku se aplikuje k pravidel. Řízení však nic neříká o tom, která pravidla to jsou. Např. při generování jazyka $L = \{ww | w \in \{a, b\}^+\}$ bychom potřebovali, aby se dvě stejné poloviny slova tvořily podle stejných pravidel. Přitom musí existovat pravidlo přidávající a i pravidlo přidávající b a v jednom kroku by se mohlo v každé půlce slova uplatnit jiné z nich. Intuitivně se tedy zdá, že tento jazyk k-gramatikou nelze generovat. Může ale existovat řešení, které není na první pohled zřejmé. Tvorba gramatik není algoritmizovatelná a bývá těžké dokázat, že jazyk nepatří do rodiny jazyků generovaných nějakým typem gramatiky.

Pro jazyk $L = \{a^{2^m} | m \geq 1\}$ bychom potřebovali v každém kroku zdvojnásobit počet neterminálů. Konstanta k je však pro celou derivaci stejná a neumožní nám přepsat v jednom kroku všechny neterminály, pokud se jejich počet zvětšuje. 2^m terminálů tak nezajistíme.

Gramatika pro jazyk $L = \{a^n b^n c^n | n \geq 1\}$ je uvedená v [HFL–96].

2.3.3 Rozptýlená kontextová gramatika

Rozptýlená kontextová gramatika se zadává stejně jako maticová bez testování výskytu. V maticové gramatice se pravidla v matici používala postupně. V rozptýlené kontextové gramatice se celá matice aplikuje najednou. V derivované větné formě musí být všechny neterminály tvořící levé strany pravidel matice. Navíc musí být tyto neterminály ve stejném pořadí. Mezi nimi se však mohou nacházet řetězce terminálů i neterminálů. Každá matice může mít libovolný počet prvků. V různých krocích se tak provádí jiný počet nahrazení (u k-gramatik byl počet předem určen).

Na první pohled možná není zřejmý rozdíl mezi sekvenčním uplatněním matice maticové gramatiky a paralelním použitím v rozptýlené kontextové. Po užití pravidla maticové gramatiky se další aplikuje na výsledek předchozího. Je-li zaveden neterminál, může tento být ihned nahrazen užitím dalšího prvku matice. V rozptýlené kontextové gramatice se celá matice aplikuje současně, a proto výsledek užití jednoho prvku nemá vliv na další prvky.

Definice 2.11 *a) Rozptýlená kontextová gramatika* (scattered context grammar) *je čtveřice* $G = (N, T, P, S)$*, kde*

- N, T, S *jsou definovány stejně jako v bezkontextových gramatikách a*
- P *je konečná množina matic*

$$(A_1 \to w_1, A_2 \to w_2, \ldots, A_k \to w_k),$$

kde $k \geq 1$, $A_i \in N$ *a* $w_i \in V_G^*$*, pro* $1 \leq i \leq k$*. Číslo* k *závisí na matici a může se pro různé matice lišit.*

b) Pro $x, y \in V_G^*$ *říkáme, že* x *přímo odvodí* y*, a značíme* $x \Rightarrow y$ *právě tehdy, když jsou splněny následující podmínky:*

$$x = x_1 A_1 x_2 A_2 \ldots x_k A_k x_{k+1}, \text{ pro nějaká } x_i \in V_G^*, 1 \leq i \leq k+1,$$
$$y = x_1 w_1 x_2 w_2 \ldots x_k w_k x_{k+1},$$
$$(A_1 \to w_1, A_2 \to w_2, \ldots, A_k \to w_k) \in P.$$

c) Jazyk $L(G)$ *generovaný rozptýlenou kontextovou gramatikou* G *je definován jako*

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

Rodinu rozptýlených kontextových gramatik označujeme λSC. Neuvažujeme-li vypouštějící pravidla, vynecháme symbol λ.

Příklad 2.19 $L = \{ww | w \in \{a, b\}^+\}$

$G = (\{S, A\}, \{a, b\}, \{m_1, m_2, m_3, m_4, m_5\}, S)$
$m_1 = (S \to AA)$ $\quad m_2 = (A \to aA, A \to aA)$
$m_3 = (A \to bA, A \to bA)$ $\quad m_4 = (A \to a, A \to a)$
$m_5 = (A \to b, A \to b)$

Na gramatice v příkladu 2.19 můžeme ukázat zmíněný rozdíl mezi rozptýlenou kontextovou gramatikou a maticovou gramatikou. Kdyby šlo o maticovou gramatiku, byla by možná následující derivace podle matice m_1:

$$vAvA \Rightarrow vaAvA \Rightarrow vaaAvA$$

Vzniklá větná forma nemusí vést ke slovu patřícímu do jazyka uvedeného v zadání příkladu. Podle rozptýlené kontextové gramatiky se celá matice použije najednou a v jednom kroku tak musí přibýt jeden terminál v každé půlce slova. A protože v jedné matici mají pravidla pravou stranu stejnou, i půlky slova vznikají stejné.

Příklad 2.20 $L = \{a^m b^n a^m | 0 \leq n \leq m\}$

$G = (\{S, A\}, \{a, b\}, \{m_1, m_2, m_3, m_4\}, S)$
$m_1 = (S \to AAA)$ $\quad m_2 = (A \to aA, A \to bA, A \to aA)$
$m_3 = (A \to aA, A \to A, A \to aA)$ $\quad m_4 = (A \to \lambda, A \to \lambda, A \to \lambda)$

Gramatika v příkladu 2.20 využívá toho, že pravidla v maticích musí být použita nejen najednou, ale i ve stejném pořadí. Tím máme zajištěno, že prostřední neterminál nám vytváří terminály b, zatímco krajní se přepisují na a. Díky matici m_3 může být symbolů b méně než a, protože b nepřidává a a ano.

2.3.4 Neuspořádaná rozptýlená kontextová gramatika

Neuspořádaná rozptýlená kontextová gramatika je podobná na předchozí rozptýlenou kontextovou gramatiku. Jediný rozdíl spočívá v tom, že není dáno pořadí pravidel v matici. Ve větné formě, na kterou se matice aplikuje, se tak mohou neterminály z levých stran pravidel nacházet v libovolném pořadí. Musí tam ovšem být všechny. Během jednoho kroku se opět aplikuje matice celá.

Definice 2.12 *a) Neuspořádaná rozptýlená kontextová gramatika* (unordered scattered context grammar) *je čtveřice $G = (N, T, P, S)$, kde N, T, S, P jsou definovány stejně jako v rozptýlené kontextové gramatice.*

b) Pro $x, y \in V_G^*$ říkáme, že x přímo odvodí y, a značíme $x \Rightarrow y$ právě tehdy, když jsou splněny následující podmínky:

$$x = x_1 A_{i_1} x_2 A_{i_2} \dots x_k A_{i_k} x_{k+1}, \text{ pro nějaká } x_j \in V_G^*, 1 \le j \le k+1,$$
$$x = x_1 w_{i_1} x_2 w_{i_2} \dots x_k w_{i_k} x_{k+1},$$
$$(A_1 \to w_1, A_2 \to w_2, \dots, A_k \to w_k) \in P,$$
$$(i_1, i_2, \dots, i_k) \text{ je permutace } (1, 2, \dots, k).$$

c) Jazyk $L(G)$ generovaný neuspořádanou rozptýlenou kontextovou gramatikou G je definován jako

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

Rodinu neuspořádaných rozptýlených kontextových gramatik označujeme λuSC. Pokud neuvažujeme vypouštějící pravidla, vynecháme symbol λ.

Rodina jazyků generovaných neuspořádanými rozptýlenými kontextovými gramatikami je podmnožinou třídy jazyků generovaných rozptýlenými. Ke každé neuspořádané se jednoduše vytvoří ekvivalentní uspořádaná rozptýlená kontextová nahrazením každé matice všemi jejími permutacemi. Potom je možné použít vždy tu novou matici, která má prvky v potřebném pořadí. Permutace prvků se tak nedělají během derivace, ale při tvorbě gramatiky.

Jsou-li všechny matice jednoprvkové, jedná se o běžnou bezkontextovou gramatiku. V každém kroku se potom totiž může užít právě jedno libovolné pravidlo. Rodina jazyků generovaných nespořádanými rozptýlenými kontextovými gramatikami tedy zahrnuje celou třídu bezkontextových jazyků. Z předchozího odstavce navíc plyne, že totéž platí i pro rozptýlené kontextové gramatiky.

Příklad 2.19 na rozptýlenou kontextovou gramatiku je současně příkladem neuspořádané kontextové gramatiky. Každá dvouprvková matice má oba prvky stejné. Je proto jisté, že na pořadí těchto pravidel nezáleží.

Naopak v příkladu 2.20 jsme pořadí prvků v maticích využili a nejde tedy o příklad neuspořádané rozptýlené kontextové gramatiky. Řešení stejného jazyka neuspořádanou verzí gramatiky je uvedeno v příkladu 2.21.

Příklad 2.21 $L = \{a^m b^n a^m | 0 \leq n \leq m\}$

$G = (\{S, A, B\}, \{a, b\}, \{m_1, m_2, m_3, m_4\}, S)$
$m_1 = (S \rightarrow ABA)$ $\quad m_2 = (A \rightarrow aA, B \rightarrow bB, A \rightarrow aA)$
$m_3 = (A \rightarrow aA, A \rightarrow aA)$ $\quad m_4 = (A \rightarrow \lambda, B \rightarrow \lambda, A \rightarrow \lambda)$

2.3.5 k-jednoduchá maticová gramatika

k-jednoduchá maticová gramatika se od bezkontextových gramatik liší nejvíce. Abeceda neterminálů je rozdělena na k samostatných, navzájem disjunktních, abeced. Tyto abecedy jsou dány v pořadí, na kterém záleží. Pravidla jsou opět prvky matic. Všechny matice musí mít stejný počet prvků. Pravidlo na n-té pozici v matici je tvořeno terminály (na pravé straně) a neterminály z n-té abecedy (na pravé i levé straně). Neterminálů na pravé straně všech pravidel jedné matice musí být stejný počet. Počty terminálů se mohou lišit.

Při aplikaci matice se ve větné formě musí vyskytovat všechny levé strany pravidel ve stejném pořadí jako v matici. Navíc z každé abecedy se v daném kroku přepisuje nejlevější neterminál. Matice se, jako ve všech gramatikách s částečným paralelismem, aplikuje celá v jednom kroku.

Počáteční neterminál nepatří do žádné z abeced a nemůže se tak nacházet na levé straně žádného pravidla. Aby jím mohla derivace začít, existuje na tvar matic i jejich použití jedna výjimka. Nachází-li se na levé straně pravidla počáteční neterminál, může být na pravé straně terminální slovo nebo k neterminálů – z každé abecedy jeden v pořadí stejném, jaké mají příslušné abecedy. Tím je zajištěno, že derivace po jednom kroku úspěšně skončí, nebo vznikne větná forma, na kterou již je možné používat pravidla výše popsaným způsobem.

Definice 2.13 *a) k-jednoduchá maticová gramatika* (k-simple matrix grammar) *je (3+k)-tice* $G = (N_1, N_2, \ldots, N_k, T, P, S)$*, kde*

- $N_1, N_2, \ldots, N_k, T$ *jsou vzájemně disjunktní abecedy (množiny* N_i*,* $1 \leq i \leq k$*, jsou množiny neterminálů, T je množina terminálů),*
- $S \notin T \cup N_1 \cup N_2 \cup \ldots N_k$*,*
- M *je konečná množina matic, které mají jeden z následujících tvarů:*

$$(S \rightarrow w), w \in T^*,$$
$$(S \rightarrow A_1 A_2 \ldots A_k),\ A_i \in N_i, 1 \leq i \leq k,$$
$$(A_1 \rightarrow w_1, A_2 \rightarrow w_2, \ldots, A_k \rightarrow w_k),$$
$$A_i \in N_i, w_i \in (N_i \cup T)^*, |w_i|_{N_i} = |w_j|_{N_j},$$
$$1 \leq i \leq k, 1 \leq j \leq k, i \neq j.$$

b) Pro $x, y \in (\{S\} \cup T \cup N_1 \cup N_2 \cup \ldots \cup N_k)^*$ *říkáme, že* x *přímo odvodí* y*, a značíme* $x \Rightarrow y$ *právě tehdy, když buď*

$$x = S \text{ a } (S \to y) \in M$$

nebo

$$\begin{aligned} &x = x_1 A_1 z_1 x_2 A_2 z_2 \ldots x_k A_k z_k,\ x_i \in T^*,\ z_i \in (T \cup N_i)^*,\ 1 \leq i \leq k, \\ &y = x_1 w_1 z_1 x_2 w_2 z_2 \ldots x_k w_k z_k, \\ &(A_1 \to w_1, A_2 \to w_2, \ldots, A_k \to w_k) \in M. \end{aligned}$$

c) Jazyk $L(G)$ *generovaný k-jednoduchou maticovou gramatikou* G *je definován jako*

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

λSM označuje rodinu k-jednoduchých maticových gramatik pro všechna přirozená k. Pokud neuvažujeme vypouštěcí pravidla, vynecháme symbol λ.

Příklad 2.22 $L = \{a^m b^n a^m | 0 \leq n \leq m\}$

$G = (\{A\}, \{B\}, \{C\}, \{a, b\}, \{m_1, m_2, m_3, m_4\}, S)$
$m_1 = (S \to ABC)$ $\quad m_2 = (A \to aA, B \to bB, C \to aC)$
$m_3 = (A \to aA, B \to B, C \to aC)$ $\quad m_4 = (A \to \lambda, B \to \lambda, C \to \lambda)$

Na příkladu 2.22 si můžeme ukázat, že podmínky dané definicí gramatiky jsou splněny. Kromě m_1, která má výjimku díky pravidlu s počátečním neterminálem na levé straně, mají všechny matice tři prvky. To odpovídá počtu množin neterminálů. Každé pravidlo obsahuje jen neterminály z množiny stejného pořadí, jaké má pravidlo v matici. Všechna pravidla v jedné matici mají na pravé straně stejný počet neterminálů. Počty terminálů v pravých stranách pravidel se v matici m_3 liší, což definici neodporuje.

Derivace podle gramatiky v příkladu 2.22 probíhá jednoduše. Po prvním kroku, který je jednoznačně dán, si volíme mezi třemi maticemi. Matice m_2 přidá současně dvojici a a jedno b, m_3 jen dvojici a a m_4 derivaci ukončí přepsáním všech neterminálů na prázdné slovo. Slovo $a^m b^n a^m$ patřící do daného jazyka je generováno v $m+2$ krocích. n-krát se užije m_2, $(m-n)$-krát m_3 a jednou m_1 a m_4.

Další příklady gramatik pro jazyky $L = \{a^n b^n c^n | n \geq 1\}$ a $L = \{ww | w \in \{a, b\}^+\}$ najdeme v [HFL–96].

3. Porovnání řízených gramatik

3.1 Generativní síla

Přidáním řízení derivace si přejeme co nejvíce zvětšit generativní sílu původně bezkontextových gramatik. V nejlepším případě můžeme získat gramatiky, které jsou i s bezkontextovými pravidly schopné generovat všechny jazyky typu 0 Chomského hierarchie. V některých případech nám stačí rodina kontextových jazyků. Již v úvodu jsme totiž zmínili, že téměř všechny člověkem uvažované jazyky jsou kontextové. Pokud řízená gramatika s bezkontextovými pravidly negeneruje ani všechny bezkontextové jazyky, je pro praktické užití většinou dost nevhodná a její význam je spíše teoretický.

Vztahy mezi třídami jazyků uvedené níže jsou převzaty z [HFL–96]. Některé jsou tam i dokázány nebo je jejich důkaz naznačen.

Z hlediska generativní síly se ukazují vhodné typy řízení předepsanými sekvencemi, které jsme definovali v části práce 2.1. Platí následující rovnosti (důkaz $\mathcal{L}(\lambda rC_{ac}) \supseteq \mathcal{L}(RE)$ je v [Kyb–85]):

$$\mathcal{L}(\lambda rC_{ac}) = \mathcal{L}(\lambda M_{ac}) = \mathcal{L}(\lambda P_{ac}) = \mathcal{L}(RE)$$

To znamená, že regulárně řízené, maticové a programované gramatiky s testováním výskytu a vypouštějícími pravidly generují všechny jazyky. Pokud tyto typy gramatik uvažujeme bez testování výskytu nebo bez vypouštějících pravidel, jejich generativní síla se zmenší.

$$\mathcal{L}(\lambda rC) = \mathcal{L}(\lambda M) = \mathcal{L}(\lambda P) \subset \mathcal{L}(RE)$$

$$\mathcal{L}(rC_{ac}) = \mathcal{L}(M_{ac}) = \mathcal{L}(P_{ac}) \subset \mathcal{L}(CS)$$

$$\mathcal{L}(rC) = \mathcal{L}(M) = \mathcal{L}(P) \subset \mathcal{L}(rC_{ac}) \quad \mathcal{L}(rC) = \mathcal{L}(M) = \mathcal{L}(P) \subset \mathcal{L}(\lambda rC)$$

Důkaz $\mathcal{L}(rC_{ac}) \subset \mathcal{L}(CS)$ je v [Bul–89]. Samotné testování výskytu bez přítomnosti vypouštěcích pravidel neumožní ani generování všech kontextových jazyků. Podle výše uvedených vztahů můžeme říct, že ke každé regulárně řízené, maticové i programované gramatice můžeme sestrojit ekvivalentní regulárně řízenou, maticovou i programovanou gramatiku. Navíc při této konstrukci nemusíme přidávat ani vypouštějící pravidla, ani testování výskytu, pokud v původní gramatice nebyly.

Jedinou výjimkou mezi gramatikami s předepsanými sekvencemi je neuspořádaná vektorová gramatika. Podle následujícího vztahu nedokáže generovat ani některé jazyky, které generují maticové gramatiky bez testování výskytu i vypouštějících pravidel. Z toho plyne, že negenerují celou třídu kontextových jazyků. Generativní síla neuspořádaných vektorových gramatik je stejná s vypouštěcími pravidly i bez nich. Následující vztahy jsou dokázány v [Kyb–85], [Pău–80], [Sti–96].

$$\mathcal{L}(CF) \subset \mathcal{L}(uV) = \mathcal{L}(\lambda uV) \subset \mathcal{L}(M)$$

Nyní se podíváme na gramatiky řízené kontextovými podmínkami, které jsme uvedli v části 2.2.

$$\mathcal{L}(\lambda C) = \mathcal{L}(\lambda sC) = \mathcal{L}(\lambda RC_{ac}) = \mathcal{L}(RE)$$
$$\mathcal{L}(C) = \mathcal{L}(sC) = \mathcal{L}(CS) \quad \mathcal{L}(RC_{ac}) \subset \mathcal{L}(CS)$$

Důkazy pro gramatiky s náhodným kontextem jsou v [Kyb–85]. Neuvažujeme-li vypouštěcí pravidla, generují podmíněné a polopodmíněné gramatiky pouze kontextové jazyky. Gramatiky s náhodným kontextem bez λ-pravidel nedokáží generovat ani některé kontextové jazyky. Přidání vypouštěcích pravidel zvětšuje generativní sílu těchto tří typů gramatik na všechny jazyky typu 0.

Do stejné skupiny gramatik s kontextovými podmínkami jsme zařadili i uspořádané gramatiky. Princip jejich řízení je ale od podmíněných, polopodmíněných i gramatik s náhodným kontextem dost odlišný. Také generativní síla se liší.

$$\mathcal{L}(O) \subseteq \mathcal{L}(\lambda O) \subset \mathcal{L}(RE)$$

Z uvedeného vztahu vidíme, že pomocí uspořádaných gramatik některé jazyky typu 0 popsat nedokážeme.

Mezi gramatikami s částečným paralelismem představenými v části 2.3 již je jen jedna, která generuje všechny jazyky typu 0. Jedná se o rozptýlenou kontextovou gramatiku.

$$\mathcal{L}(M) = \mathcal{L}(uSC) \subset \mathcal{L}(SC) \subseteq \mathcal{L}(CS)$$
$$\mathcal{L}(\lambda M) = \mathcal{L}(\lambda uSC) \subset \mathcal{L}(\lambda SC) = \mathcal{L}(RE)$$

Důkaz $\mathcal{L}(RE) \subseteq \mathcal{L}(\lambda SC)$ je v [Act–89], [Act–95] a [Bul–95]. Podle uvedených vztahů je neuspořádaná rozptýlená gramatika generativně slabší než odpovídající verze uspořádané rozptýlené gramatiky. Naopak má stejnou generativní sílu jako maticová gramatika bez testování výskytu (a tedy i programovaná a regulárně řízená bez testování výskytu).

Ze všech typů řízených gramatik uvedených v této práci jen indické paralelní gramatiky nedokáží generovat některé bezkontextové jazyky. $\mathcal{L}(CF)$ a $\mathcal{L}(IP)$ jsou totiž neporovnatelné. Existují kontextové jazyky, které je možné popsat indickou paralelní gramatikou, ale jiné bezkontextové jí popsat nelze. Generativní síla indických paralelních gramatik je stejná, uvažujeme-li vypouštěcí pravidla, nebo ne. Důkaz tohoto tvrzení je uveden v [IAT–79].

$$\mathcal{L}(kG) \subseteq \mathcal{L}(M) \quad \mathcal{L}(\lambda kG) \subseteq \mathcal{L}(\lambda M)$$

Třída jazyků generovaných k-gramatikami je porovnávána s třídou jazyků generovaných maticovými gramatikami. Spolu s dřívějšími vztahy vyplývá, že k-gramatiky bez λ-pravidel generují vlastní podmnožinu kontextových jazyků a s vypouštěcími pravidly podmnožinu obecných jazyků (typu 0 Chomského hierarchie).

Posledním typem řízených gramatik, které jsme uvedli v přehledu, byly k-jednoduché maticové gramatiky. Ty však také nedokáží popsat některé kontextové jazyky.

$$\mathcal{L}(SM) \subset \mathcal{L}(\lambda SM) \subset \mathcal{L}(CS)$$

V úvodu této části práce jsme určili za nejvhodnější takové typy řízení, které nám zajistí generování všech jazyků typu 0. Tuto podmínku splňují, a z hlediska generativní síly se tedy zdají nejvhodnější, tyto typy řízených gramatik:

- regulárně řízené s testováním výskytu
- maticové s testováním výskytu
- programované s testováním výskytu
- podmíněné
- polopodmíněné
- gramatiky s náhodným kontextem
- rozptýlené kontextové

3.2 Algoritmus derivace

Různé typy řízení uvedené v této práci se mezi sebou hodně liší. Díky tomu se musí lišit i algoritmy, které provádějí derivaci podle řízených gramatik. Porovnáme je podle toho, co je nutno naprogramovat (jak je obtížné vytvořit algoritmus pro derivaci), a zároveň, co musí provádět počítač v každém kroku derivace. Nepůjde o přesné určení složitosti algoritmu, spíše intuitivní vyjmenování operací nutných k určení aplikovatelných pravidel.

3.2.1 Klasické neřízené gramatiky

Jako základ pro porovnání nám poslouží klasické neřízené gramatiky podle Chomského hierarchie. V nich je použitelné takové pravidlo, jehož levá strana se vyskytuje v aktuální větné formě. Jiná podmínka neexistuje. V každém kroku jsou tedy procházena všechna pravidla a levá strana každého z nich je hledána v aktuální větné formě. Pokud je nalezena, pravidlo je přidáno mezi použitelná a pokračuje se dalším pravidlem. Vyhledávání je jednodušší v případě bezkontextových gramatik, protože levé strany tvoří pouze jeden neterminál. Jde vlastně o hledání znaku v řetězci. Kontextové a obecné gramatiky již vyžadují hledání podřetězce.

Pokud je množina použitelných pravidel prázdná, je derivace zablokovaná. Může to být i úspěšný konec derivace, kdy větnou formu tvoří jen terminály. Optimální je situace, kdy je množina použitelných pravidel jednoprvková. Potom může být pravidlo aplikováno automaticky a pokračuje se hledáním množiny aplikovatelných pravidel pro další krok. Poslední možností je víceprvková množina. Zde nastává okamžik nedeterminismu. Pokračování derivace vyžaduje výběr pravidla uživatelem nebo nějakým pseudonáhodným generátorem.

Jiný druh nedeterminismu nastává, vyskytuje-li se levá strana pravidla ve větné formě vícekrát. Pro pokračování derivace se musí rozhodnout, který výskyt má být přepsán. Při automatickém provádění derivace jsou nedeterminismy velkou nevýhodou. Většinou si strojově prováděnou činností chceme zjednodušit práci. Pokud musíme často volit pravidlo pro pokračování, je to stejné, jako bychom derivaci prováděli sami. Proto budeme na počet nedeterminismů v této části práce klást velký důraz.

V bezkontextových gramatikách často uvažujeme levou derivaci. Vždy je nějakým pravidlem přepsán nejlevější neterminál. Tím nám odpadá nedeterminismus s volbou výskytu levé strany. Ulehčuje se také výběr použitelných pravidel. Ve větné formě nalezneme nejlevější neterminál a vybereme pravidla, která jej mají na levé straně. Takto vybraných pravidel je obecně méně než těch, jejichž levá strana se vyskytuje kdekoliv ve větné formě. Proto i nedeterminismus ve volbě pravidla nastává méně často.

Kontextové gramatiky pro některé jazyky, které používáme jako příklady v kapitole 2, najdeme v [Sal–73]. Dvě z nich budou později uvedeny i v této práci jako příklady 3.2 a 3.3. Na nich si můžeme všimnout, že kontextové gramatiky často potřebují, ve srovnání s řízenými, mnohem více kroků na vygenerování stejného slova. Tím je i potřebných operací počítače při simulaci více. Levé strany kontextových pravidel mohou tvořit řetězce terminálů a neterminálů a jejich vyhledávání ve větné formě je složitější, než nalezení pouze samotných neterminálů z levých stran pravidel řízených gramatik. Použití vhodné řízené gramatiky by tedy mělo ušetřit operace počítače proti kontextové gramatice generující stejný jazyk.

3.2.2 Řízené gramatiky

Podobně jako v klasických gramatikách, i v řízených se musí levá strana pravidla nacházet v derivované větné formě. Při výběru použitelného pravidla však musí být splněna ještě nějaká další podmínka. Protože je odlišná v jednotlivých typech řízení, je její ověření různě složité na naprogramování i na výpočet. Nedeterminismů v průběhu derivace jsou dva druhy jako v klasických neřízených gramatikách. V daném kroku může být použitelných více pravidel a levá strana vybraného může být obsažena ve větné formě vícekrát. V neřízených gramatikách je rozhodování mezi pravidly časté. Použitelná mohou být totiž všechna, jejichž levá strana se ve větné formě vyskytuje. Řízené gramatiky množství použitelných pravidel omezují a nutnost volby pravidla nastává méně často.

Leftmost derivace

V klasických gramatikách je výhodné použití levé derivace. Usnadňuje hledání použitelných pravidel a zmenšuje počet nedeterminismů. Definice řízených gramatik nám použití levé derivace takové, jaká byla navržena pro klasické gramatiky, neumožňuje. Řízení může způsobit, že na nejlevější neterminál ve větné formě není možné použít žádné pravidlo. Až po aplikaci několika pravidel na jiné neterminály v různých místech větné formy nastane situace, kdy je možné přepsat ten nejlevější.

[Sal–73] zavádí pro maticové gramatiky jiný mechanismus výběru výskytu levé strany pro aplikaci pravidla, jehož podstata je podobná. Anglicky se nazývá „leftmost restriction“. Česky by mohl znít jako „nejlevější omezení“, ale v dalším textu budeme používat pojem leftmost omezení nebo leftmost derivace.

Leftmost derivace vždy vybírá pro přepsání nejlevější neterminál, na který lze nějakou matici použít. Toto omezení platí vždy jen pro první pravidlo v matici a další jsou již používána způsobem, který byl pro maticové gramatiky definován. Při použití leftmost derivace se podle definice nepoužívá testování výskytu.

Nabízí se možnost používat princip leftmost derivace i v ostatních typech gramatik. Z aplikovatelných pravidel by se v každém kroku vybralo to, jehož levá strana se ve větné formě vyskytuje nejvíce vlevo. Jen pokud je více pravidel se stejnou levou stranou, je potřeba pokračování zvolit. Nedeterminismus s volbou výskytu levé strany, na který se má pravidlo užít, bychom tak odstranili. Také počet pravidel aplikovatelných v daném kroku by byl menší a častěji by bylo pokračování jednoznačné.

Při leftmost derivaci jsme zatím neuvažovali použití pravidel v módu testování výskytu. Nejjednodušší je definovat, že pravidlo ve smyslu testování výskytu je možno použít právě tehdy, když nejde aplikovat žádné pravidlo klasickým způsobem.

Použití leftmost omezení zjednodušuje automatické provádění derivace. Zmenšuje se počet nutných zásahů uživatele. Důležité ale je, aby se leftmost omezením nesnížila generativní síla gramatik. Existují gramatiky, které generují s tímto omezením jiný jazyk než bez něj. Gramatika v příkladu 3.1 generuje podle klasické definice jazyk $L = \{a^n b^n c^n | n \geq 1\}$ a leftmost derivací pouze slovo abc. Pravidlo f_5 má totiž přednost před f_4, protože přepíše neterminál více vlevo.

Příklad 3.1

$G = (\{S, A, B, C\}, \{a, b, c\}, \{f_1, \ldots, f_7\}, S)$

$f_1 = (S \to ABC, \{f_4, f_5\}, \phi)$ $f_2 = (A \to aA, \{f_4, f_5\}, \phi)$ $f_3 = (B \to bB, \{f_2\}, \phi)$

$f_4 = (C \to cC, \{f_3\}, \phi)$ $f_5 = (A \to a, \{f_6\}, \phi)$ $f_6 = (B \to b, \{f_7\}, \phi)$

$f_7 = (C \to c, \phi, \phi)$

Existence gramatik, které generují leftmost derivací jiný jazyk než klasickou, však neznamená, že leftmost omezení sníží generativní schopnost řízených gramatik. Ke každé gramatice může existovat jiná, která generuje stejný jazyk leftmost derivací. Generativní síla by v takovém případě byla s leftmost omezením stejná jako bez něj. V [Sal–73] je dokázáno, že maticové gramatiky s leftmost derivací generují i bez testování výskytu všechny jazyky typu 0 Chomského hierarchie. Přidání leftmost omezení tedy generativní sílu zvětší. Pro ostatní typy řízených gramatik zatím podobný důkaz není.

Leftmost derivaci si nyní definujeme ještě jinak. Aplikovatelná pravidla se určují běžným způsobem a vybrané se aplikuje na nejlevější výskyt levé strany. Na rozdíl od předchozí definice se tedy neomezuje výběr pravidel. Odstraní se tak jen nedeterminismus při volbě výskytu, ale aplikovatelných pravidel je stejně jako při běžné derivaci. Pro gramatiky, ve kterých výběr pravidla není řízen tvarem aktuální větné formy, se zdá zřejmé, že použití druhé varianty leftmost omezení generativní sílu nezmění. Podle definice derivace musí pro každé slovo existovat nějaká sekvence pravidel, která vede k jeho vygenerování. Pomocí leftmost omezení změníme jen pořadí, ve kterém se aplikují pravidla na jednotlivé výskyty neterminálů ve větné formě.

Trochu odlišná situace je v případě gramatik řízených kontextovými podmínkami. Přepsání levého výskytu neterminálu může způsobit, že podmínka žádného pravidla nebude splněna. Při přepsání pravého výskytu by však derivace pokračovat mohla. Pokud pro každý jazyk generovaný klasicky existuje gramatika, která jej generuje leftmost derivací, nesníží leftmost omezení generativní schopnost jazyka. V takovém případě by bylo vhodné leftmost omezení používat, protože menší počet nedeterminismů je při automatickém průběhu derivace výhodný. Pro člověka je také přirozenější, když se větná forma přepisuje, a terminální slovo vzniká, zleva.

Konstrukce a běh algoritmu

V regulárně řízených gramatikách je pořadí pravidel při derivaci určeno regulárním výrazem. Tento výraz musí být zpracován. Nejjednodušší je převést jej na začátku derivace na nějak reprezentovaný konečný automat, který bude v každém kroku určovat použitelná pravidla. Z nich se vyberou ta, jejichž levá strana je obsažena ve větné formě, nebo ta, která jsou současně i v množině pravidel použitelných v módu testování výskytu. Podle aplikovaného pravidla se převede řídící konečný automat do dalšího stavu. Při konstrukci programu provádějícího derivaci musíme sestrojit algoritmus, který převod regulárního výrazu na konečný automat zrealizuje. Reprezentace automatu by měla být taková, aby se použitelná pravidla určovala co nejjednodušeji. Nedeterminismus ve volbě pravidla vzniká, když je v regulárním výrazu možnost $0-n$ opakování nějaké sekvence nebo volba z více sekvencí pravidel. V konečném automatu tomu odpovídá více možných přechodů ze současného stavu do jiných. Pokračování musí vybrat uživatel.

Algoritmus provádějící derivaci podle maticových gramatik si musí pamatovat rozdělení do matic. Nejjednodušší je asi uložení pravidel v dvourozměrném poli. Dále si musí pamatovat pravidla, která je možno použít v módu testování výskytu. To lze vyřešit například nějakou proměnnou přidanou do struktury pravidla nebo samostatným polem pravidel použitelných ve smyslu testování výskytu. Při derivaci nastává nedeterminismus jen při výběru matic. Vždy po aplikaci prvního pravidla se automaticky použijí postupně všechna pravidla vybrané matice. V těchto automatických krocích se nemusí určovat použitelná pravidla a ušetří se operace počítače.

Maticové a regulárně řízené gramatiky používají pravidla v módu testování výskytu. Podle definice takto lze užít jen některá pravidla. Ta je nutno před derivací zadat a počítač si je musí pamatovat. Pokud není levá strana obsažena ve větné formě, musí se zjistit, jestli je toto pravidlo možné užít také ve smyslu testování výskytu. Jednodušší by bylo, kdyby tak mohla být aplikována všechna pravidla. Simulátor by si nemusel množinu pamatovat a prohledávat ji před použitím pravidel. Kdybychom chtěli takto změněnou definici používat, museli bychom dokázat, že nesníží generativní sílu gramatik. Snadnější provádění derivace by nevyvážilo to, že některé jazyky nemůžeme gramatikou popsat.

Při simulaci programovaných gramatik musí být pravidlům přiřazeny dvě množiny pravidel pro další krok. V každém kroku, kromě prvního, musí navíc algoritmus vědět, které pravidlo bylo použito naposledy a jakým způsobem. Pokud šlo o klasickou aplikaci, hledá aktuálně použitelná pravidla v poli úspěchu, jinak v poli neúspěchu. Nemusí se tak prohledávat všechna pravidla a provedených operací je méně. I počet nedeterminismů je omezen. Při konstrukci algoritmu je nutné jen navrhnout vhodnou reprezentaci pole úspěchu a neúspěchu, aby v nich nebylo složité nalezení použitelných pravidel. Nabízí se například pole ukazatelů na strukturu reprezentující pravidlo, pokud to programovací jazyk umožňuje.

Algoritmus simulace vektorových gramatik již je trochu složitější. Jako v maticových gramatikách musí být pravidla uložena tak, aby se poznalo, která patří do stejné matice. Po aplikaci jednoho prvku matice musí algoritmus zajistit, aby se někdy použily i ostatní. Asi nejlepším řešením je přiřadit každému pravidlu čítač, kolikrát toto pravidlo ještě musí být užito. Po aplikaci pravidla s hodnotou čítače nula se všem ostatním ze stejné matice čítač inkrementuje (i ostatní prvky matice musí být v budoucnu užity). Při použití pravidla s hodnotou čítače nenulovou se jeho hodnota dekrementuje. Derivace může skončit, jsou-li hodnoty všech čítačů rovny nule.

Použitelná pravidla se hledají mezi všemi prvky matic. Počet operací na toto hledání je tak stejný jako v případě klasických gramatik. I nedeterminismů zůstává stejný počet jako v neřízených gramatikách, protože výběr použitelných pravidel nic neomezuje.

Podmíněné, polopodmíněné a gramatiky s náhodným kontextem mají společné vlastnosti. S každým pravidlem si musí program provádějící derivaci pamatovat ještě něco dalšího. V podmíněné gramatice je to regulární výraz, v polopodmíněné a gramatice s náhodným kontextem dvě množiny. V každém kroku se použitelná pravidla hledají mezi všemi. Pro každé pravidlo, jehož levá strana se vyskytuje ve větné formě, se musí ověřit ještě podmínka spojená s řízením. V podmíněných gramatikách je nutné sestrojit algoritmus rozhodující, zda větná forma vyhovuje regulárnímu výrazu. Polopodmíněné a gramatiky s náhodným kontextem vyžadují ověření, jestli se všechny prvky povoleného kontextu nacházejí v aktuální větné formě a všechny prvky zakázaného kontextu se v ní nevyskytují. Takové zkoumání větné formy u všech tří typů zvyšuje výpočetní složitost proti klasickým gramatikám. Díky podmínkám na větnou formu se ale snižuje počet nedeterminismů při volbě pravidel, protože ne všechna podmínce vyhoví.

Algoritmus derivace podle uspořádaných gramatik musí mít kromě pravidel k dispozici i jejich uspořádaní. Reprezentace může být různá. Například může mít každé pravidlo přiřazeno pole označení pravidel, která jsou prioritnější (větší podle uspořádání). Pokud bychom jen přiřadili pravidlům přirozená čísla, jejichž uspořádání by odpovídalo prioritám pravidel, nastal by problém s vyjádřením neporovnatelných pravidel. V každém kroku derivace se klasickým způsobem hledají aplikovatelná pravidla. Potom musí být mezi nimi nalezena nějakým vhodným algoritmem maximální podle uspořádání. Počet aplikovatelných pravidel se omezí a tím i počet nedeterminismů.

V derivaci podle indické paralelní gramatiky se aplikuje pravidlo na všechny výskyty levé strany současně. Tím je odstraněn jeden typ nedeterminismu a nepotřebujeme ani leftmost omezení. Jinak výběr použitelných pravidel probíhá stejně jako v klasických gramatikách. Algoritmus derivace je tedy téměř stejný jako pro bezkontextové gramatiky, jen přepis podle pravidla se liší, protože jsou pravou stranou nahrazeny všechny výskyty levé strany.

Při derivaci podle k-gramatiky si musí počítač pamatovat navíc jen jedno číslo. Je jím konstanta k. V každém kroku se použitelná pravidla hledají stejně jako v klasických gramatikách. Z nich jich musí být zvoleno právě k a ta jsou aplikována současně. Nedeterminismus ve volbě pravidla se tak rozšiřuje na volbu více pravidel. Navíc i jejich levé strany mohou být ve větné formě vícekrát a pro každé pravidlo je nutné vybrat jeden konkrétní výskyt k přepsání. Ani nutnost použití k pravidel současně nevylučuje možnost leftmost derivace a tím omezení počtu nedeterminismů při volbě výskytu. Pro aplikaci by se vybralo k takových pravidel, jejichž levé strany jsou ve větné formě obsaženy co nejvíce vlevo. Podle druhé definice leftmost derivace by se určilo k libovolných pravidel a ta by se aplikovala na nejlevější výskyty svých levých stran. Pokud by mezi nimi byla nějaká dvě pravidla se stejnou levou stranou, musel by uživatel rozhodnout, které se aplikuje více vlevo.

Při derivaci podle rozptýlené kontextové gramatiky se nehledají použitelná pravidla ale celé matice. Najdeme první výskyt levé strany prvního pravidla ve větné formě. Levou stranu dalšího pravidla již hledáme jen ve zbytku větné formy. Tak to opakujeme pro všechna pravidla v matici. Jestliže se podaří nalézt všechny levé strany, máme zajištěno i to, že jsou v potřebném pořadí a matice je použitelná. Postupně projdeme všechny matice. Je-li jich použitelných více, musí

vybrat pokračování uživatel. Aplikovatelnost celé matice se tedy určí při jednom průchodu větnou formou i maticí. Vybraná matice se potom aplikuje celá současně.

I při derivaci podle neuspořádané rozptýlené kontextové gramatice se hledají použitelné matice. Levá strana každého pravidla v matici se může vyskytovat kdekoliv ve větné formě. Při ověřování použitelnosti je tedy pro každé pravidlo nutné projít větnou formu, nebo pro každý neterminál ve větné formě levé strany všech pravidel v matici. Při druhé možnosti by algoritmus musel označit pravidla, jejichž levá strana byla již nalezena. Bylo by ale zajištěno, že pro pravidla se stejnou levou stranou se tato vyskytuje ve větné formě v dostatečném počtu. V první verzi algoritmu to musíme zajistit nějak jinak. Nedeterminismus může nastat i v pořadí, v jakém se mají pravidla jedné matice použít. Jinak je simulace stejná jako v uspořádané verzi rozptýlených kontextových gramatik.

Při derivaci podle k-jednoduché maticové gramatiky jsou také pravidla v maticích. V každém kroku musíme najít použitelnou matici. Větná forma se vždy dá rozdělit na k částí a levá strana každého pravidla matice se musí nacházet jako první neterminál v příslušné části. Je-li toto splněno, je matice aplikovatelná. Při implementaci musíme jenom vymyslet algoritmus, který rozdělí větnou formu na zmiňované části. Například, když vydělíme počet neterminálu ve větné formě číslem k, víme, kolik jich je v každé části. Terminály nás při hledání výskytů levých stran nezajímají. V k-jednoduchých maticových gramatikách neexistuje nedeterminismus s volbou výskytu levé strany ve větné formě, protože je podle definice vždy přepsán nejlevější neterminál v každé části.

Z uvedených typů gramatik jsou z hlediska implementace simulátoru derivace výhodné programované a maticové gramatiky. V porovnání s klasickými gramatikami nevyžadují navíc žádné složité algoritmy. Operací při určování použitelných pravidel je méně, protože se nevybírá ze všech. Nedeterminismy nastávají méně často, protože někdy je pravidlo díky řízení určeno jednoznačně. Při automatické derivaci proto nemusí uživatel tak často rozhodovat o pokračování.

Snadno implementovatelná je i indická gramatika. Její simulátor je hodně podobný simulátoru bezkontextové gramatiky a derivace probíhá v méně krocích. Implementace rozptýlené kontextové gramatiky se od klasických gramatik liší, ale není těžké ji navrhnout. Nalezení aplikovatelné matice je přibližně stejně náročné jako nalezení použitelného pravidla v klasických gramatikách. Derivace ale proběhne v méně krocích, protože je pravidel současně užito více.

Ostatní typy řízených gramatik již vyžadují implementaci složitějších algoritmů, provádějí více operací v každém kroku nebo obsahují hodně nedeterminismů.

3.3 Přirozenost návrhu gramatiky

3.3.1 Porovnání řízených a neřízených gramatik

V části porovnávající gramatiky podle generativní síly jsme uvedli, že některé typy řízení zajistí i s použitím bezkontextových pravidel generování jazyků typu 0 nebo alespoň 1 podle Chomského hierarchie. Mohlo by se ale zdát, že řízené gramatiky nejsou potřeba. Všechny jazyky přece můžeme generovat obecnými gramatikami a velkou část i kontextovými. Již dříve jsme zmínili výhody řízených gramatik při automatickém provádění derivace. Dalším důvodem, proč jsou

vhodnější než klasické, je přirozenost jejich návrhu. Sestrojení neřízené gramatiky pro kontextový jazyk vyžaduje většinou nějaké pomocné neterminály a pravidla, která zajistí správný postup při derivaci. Jejich význam nemusí být na první pohled zřejmý. Z výsledné gramatiky potom není snadné poznat, jaký jazyk generuje. Často je i problém dokázat, že opravdu výsledný jazyk odpovídá požadovanému. Navíc bezkontextová pravidla v řízených gramatikách se navrhují snadněji než kontextová, která jsou nutná v klasických gramatikách.

Ukážeme si dva příklady obecných gramatik řešících jazyky, které jsme užívali i pro ukázku schopností řízených gramatik. Oba příklady jsou převzaty z [Sal–73].

Příklad 3.2 $L = \{a^n b^n c^n | n \geq 1\}$

$G = (\{S, X, Y\}, \{a, b, c\}, \{f_1, \ldots, f_7\}, S)$
$f_1 = S \to abc \qquad f_2 = S \to aXbc \qquad f_3 = Xb \to bX$
$f_4 = Xc \to Ybcc \qquad f_5 = bY \to Yb \qquad f_6 = aY \to aaX$
$f_7 = aY \to aa$

Pravidlo f_1 je potřeba, protože pomocí dalších dokážeme generovat $a^n b^n c^n$ jen pro $n \geq 2$. Postup derivace si popíšeme v jednotlivých krocích. Po použití pravidla f_2 se dostaneme do situace a) s $n = 1$.

a) $a^n X b^n c^n \Rightarrow^*_{f_3} a^n b^n X c^n$

b) $a^n b^n X c^n \Rightarrow_{f_4} a^n b^n Y b c^{n+1}$

c) $a^n b^n Y b c^{n+1} \Rightarrow^*_{f_5} a^n Y b^{n+1} c^{n+1}$

d) $a^n Y b^{n+1} c^{n+1} \Rightarrow^*_{f_7} a^{n+1} b^{n+1} c^{n+1}$ a derivace končí nebo
$a^n Y b^{n+1} c^{n+1} \Rightarrow^*_{f_6} a^{n+1} X b^{n+1} c^{n+1}$ a pokračuje se znovu bodem a).

Když se ve větné formě vyskytuje X, je všech terminálů stejný počet. Neterminál Y znamená, že terminálů b a c je o jeden více než a. Přidání jednoho výskytu všech terminálů způsobí, že musí postupně X přejít až za všechna b a potom Y zase před ně.

V příkladu 2.6 jsme stejný jazyk generovali programovanou gramatikou. Jednoduše jsme zajistili, že se ve třech krocích po sobě přidal jeden výskyt tří terminálů. Je to přirozenější, než vymýšlet neterminály putující větnou formou tam a zpět. V [Sal–73] jsou uvedeny ještě další čtyři obecné generativní gramatiky generující stejný jazyk. Způsob, kterým zajišťují stejný počet různých terminálů, je v nich ještě nepřehlednější.

Příklad 3.3 $L = \{ww | w \in \{a, b\}^*\}$

$G = (\{S, X, Y, Z, A, B\}, \{a, b\}, \{f_1, \ldots, f_{11}\}, S)$
$f_1 = S \to XYZ \qquad f_2 = XY \to aXA \qquad f_3 = XY \to bXB \qquad f_4 = Ab \to bA$
$f_5 = Ba \to aB \qquad f_6 = Aa \to aA \qquad f_7 = Bb \to bB \qquad f_8 = BZ \to YbZ$
$f_9 = AZ \to YaZ \qquad f_{10} = aY \to Ya \qquad f_{11} = bY \to Yb \qquad f_{12} = XY \to \lambda$
$f_{13} = Z \to \lambda$

Najít v těchto pravidlech derivaci slova patřícího do daného jazyka není jednoduché. Proto si ji stručně popíšeme.

Derivace začíná následovně:

$$S \Rightarrow_{f_1} XYZ \Rightarrow_{f_2} aXAZ \Rightarrow_{f_9} aXYaZ, \text{ nebo}$$

$$S \Rightarrow_{f_1} XYZ \Rightarrow_{f_3} bXBZ \Rightarrow_{f_8} bXYbZ$$

Tak získáme větnou formu, která je na začátku dalšího kousku derivace. Tento kousek se opakuje, dokud se nevygeneruje větná forma obsahující celé budoucí slovo a tři neterminály navíc.

$$vXYvZ \Rightarrow_{f_2} vaXAvZ \Rightarrow^*_{f_4,f_6} vaXvAZ \Rightarrow_{f_9} vaXvYaZ \Rightarrow^*_{f_{10},f_{11}} vaXYvaZ, \text{ nebo}$$

$$vXYvZ \Rightarrow_{f_3} vbXBvZ \Rightarrow^*_{f_5,f_7} vbXvBZ \Rightarrow_{f_8} vbXvYbZ \Rightarrow^*_{f_{10},f_{11}} vbXYvbZ$$

Derivace může skončit kdykoliv následovně

$$vXYvZ \Rightarrow_{f_{12}} vvZ \Rightarrow_{f_{13}} vv$$

V příkladech 2.3 a 2.8 jsme téměř stejný jazyk generovali regulárně řízenou a programovanou gramatikou. Princip derivace v nich byl jednoduchý. Vygenerovalo se tolik stejných neterminálů, kolikrát se má slovo opakovat (v tomto konkrétním případě dva). Potom jsou vždy všechny postupně přepsány podle stejného pravidla, a tím do všech opakování přibude stejný terminál. Takový postup se opakuje tolikrát, kolikrát je potřeba. Existuje ještě přirozenější řešení pro programované i regulárně řízené gramatiky bez testování výskytu. Stačí vygenerovat dva za sebou jdoucí různé neterminály. Přepis prvního si potom vždy vynutí přepis druhého podle ekvivalentního pravidla. Toto řešení programovanou gramatikou vypadá takto:

$G = (\{S, A, B\}, \{a, b\}, \{f_1, \ldots, f_7\}, S)$

$f_1 = (S \to AB, \{f_2, f_3, f_6\}, \phi)$ $f_2 = (A \to aA, \{f_4\}, \phi)$ $f_3 = (A \to bA, \{f_5\}, \phi)$

$f_4 = (B \to aB, \{f_2, f_3, f_6\}, \phi)$ $f_5 = (B \to bB, \{f_2, f_3, f_6\}, \phi)$ $f_6 = (A \to \lambda, \{f_7\}, \phi)$

$f_7 = (B \to \lambda, \phi, \phi)$

Pro velkou část kontextových jazyků je sestrojení gramatiky obtížné a nestačí převést intuitivní představu o průběhu derivace na gramatiku. Některé typy řízených gramatik umožňují určit přímo pořadí pravidel. Díky tomu můžeme gramatiku sestrojit přesně podle toho, jak si představujeme, že by mohla probíhat derivace. Nemusíme vymýšlet žádné pomocné nepřirozené obraty, které by nám konkrétní derivaci zajistily. Proto jsou některé řízené gramatiky vhodnější než klasické. Neplatí to ovšem pro všechny typy řízení. Jejich porovnání z hlediska přirozenosti návrhu gramatiky pro jazyk následuje v další sekci této práce.

3.3.2 Porovnání typů řízených gramatik

Když uvažujeme nějaký jazyk a chceme pro něj sestrojit gramatiku, je přirozené vymyslet, jak by mohla probíhat derivace. Problém je potom zajistit, aby opravdu byla pravidla aplikována v požadovaném pořadí. Vymýšlení dalších pravidel, pomocných symbolů a umělých obratů, které promyšlený průběh derivace vynutí, návrh gramatiky komplikuje.

Řízení programovaných, maticových a regulárně řízených gramatik spočívá v definování sekvencí pravidel. To nám umožní vynutit si derivaci podle naší představy. Při konstrukci maticové gramatiky si v derivaci najdeme sekvence, které proběhnou vždy celé, a uděláme z nich matice. Musíme je najít tak, aby se derivace větvila pro různá slova při volbě matic. Matice mají ale konečný počet prvků. Pokud si přejeme přepsat předem neznámý počet stejných neterminálů současně, nestačí nám vytvořit matici, která takový přepis provede. Označme pro snadnější vysvětlení takový neterminál X. Jestliže po aplikaci jedné matice zůstanou některá X nepřepsána, v dalším kroku by mohla derivace pokračovat libovolnou maticí a jejich přepis by nebyl vynucen. V takové situaci se používá pravidlo v módu testování výskytu. Umístíme ho na začátek matic, které mohou být aplikovány až po přepisu všech X. Jeho princip spočívá v převedení jednoho z X na pomocný, nepřepsatelný neterminál, který derivaci zablokuje. Proto je matice použitelná, až se ve větné formě nevyskytuje žádné X. Tehdy se pomocné pravidlo použije ve smyslu testování výskytu a pokračuje se dalším pravidlem matice. Tímto obratem zajistíme derivaci, kterou si přejeme. Pokud jej neznáme, je vytvoření gramatiky nesnadné.

Regulárně řízené gramatiky mají pořadí pravidel určeno regulárním výrazem. Ten nám umožňuje zadat opakování i volbu sekvencí. Zdálo by se, že převést představu o průběhu derivace na regulárně řízenou gramatiku je jednoduché. Problémem je zajištění potřebného počtu opakování nějaké sekvence. Používá se na to podobný obrat, jaký jsme uvažovali v maticových gramatikách. Navrhneme pravidla tak, aby po potřebném počtu opakování cyklu ve větné formě nebyl žádný výskyt nějakého konkrétního neterminálu X. V regulárním výrazu dáme za takovou opakovanou sekvenci pravidlo, které neterminál X převede na jiný, neodstranitelný. Pokud cyklus proběhne tolikrát, kolikrát si přejeme, pomocné pravidlo se použije ve smyslu testování výskytu. Derivace pokračuje dál. V případech, kdy bude opakování cyklu méně, derivace nevede k terminálnímu slovu. Nevýhodou při návrhu může být také samotný regulární výraz. Pro složitější gramatiky s velkým počtem pravidel je i výraz složitý a není snadné jej vytvořit tak, aby odpovídal všem možným průběhům derivace.

Programované gramatiky umožňují převod představy o průběhu derivace na gramatiku snadno. Pomocí pole úspěchu můžeme určit pravidlo pro následující krok. Tím si vynutíme sekvence. Potřebujeme-li opakování sekvence, do pole úspěchu posledního pravidla této posloupnosti dáme opět první pravidlo. Navíc díky poli neúspěchu můžeme provádět podmíněná rozhodnutí o pokračování na základě vzhledu větné formy. Když je v některém poli více pravidel, získáme v derivaci nedeterminismus. Můžeme tak zajistit derivaci více různých terminálních slov na základě rozhodnutí o pokračování. Máme tedy k dispozici podobné konstrukce, jaké jsou ve většině programovacích jazyků. Tvorba programované gramatiky je hodně podobná konstrukci algoritmu. Většinou nepotřebujeme žádná pomocná pravidla ani symboly. Stačí vymyslet přesně průběh derivace a vyjádřit jej pomocí pravidel gramatiky včetně polí úspěchu a neúspěchu.

Pro programované gramatiky si můžeme zavést podprogramy. Podprogram je skupina pravidel, která tvoří nějaký logický celek. Definice podprogramu může být různá. Jedna je používána v [Sal–73]. My si zavedeme jinou, která bude užita i v části s praktickým příkladem.

V polích úspěchu a neúspěchu pravidel podprogramu jsou pouze pravidla této skupiny, nebo slovo out znamenající konec podprogramu. U slova out uvedeme symbol φ nebo σ, kterým určíme pole pro pokračování po skončení podprogramu. Některá pravidla jsou označena za počáteční například šipkou. Je-li počátečních více, je nedeterminismus již na začátku podprogramu. Použití

podprogramu v těle jiných se chová stejně jako jedno pravidlo. Místo samotného znění pravidla bude název podprogramu, v poli úspěchu budou pravidla, kterými se bude pokračovat po dosažení $out\ \sigma$, a v poli neúspěchu pravidla pro pokračování po $out\ \varphi$. Takové podprogramy nám zajistí přehlednost při návrhu gramatiky. Některé podprogramy mohou být použity v gramatikách pro různé jazyky a tím nám usnadní jejich návrh. Přitom můžeme pravidla podprogramu kdykoliv spojit dohromady a vytvořit tak programovanou gramatiku odpovídající původní definici.

Návrh neuspořádané kontextové gramatiky pro konkrétní jazyk je složitější, než návrh maticové gramatiky. Její řízení spočívá jen ve vynucení stejného počtu aplikací různých pravidel. Musíme vymyslet derivaci a gramatiku pro ni tak, aby na pořadí pravidel nezáleželo. Vzhledem ke generativní síle se nám to pro některé jazyky nepovede.

Podmíněné, polopodmíněné a gramatiky s náhodným kontextem neumožňují přímo vynutit požadované pořadí pravidel. Musíme derivaci vymyslet tak, aby se mohlo pravidlo vybírat podle větné formy. Podmíněné gramatiky přiřazují každému pravidlu regulární výraz popisující celou větnou formu. To nám dává možnost určit přesně, jak má forma vypadat před aplikací pravidla. Navrhnout regulární výrazy tak, aby bylo pravidlo použitelné právě tehdy, kdy požadujeme, není jednoduché. Již pro triviální jazyk v příkladu 2.10 jsou regulární výrazy hodně složité. Pro rozsáhlejší jazyky (např. programovací) je velmi těžké vytvořit výrazy tak, aby byla gramatikou generována všechna slova jazyka (všechny možné zdrojové kódy jazyka).

Použitelnost pravidel polopodmíněné gramatiky je dána výskytem podslov ve větné formě. Nemusíme tedy popsat celou větnou formu, která může během derivace obsahovat velký počet symbolů. Přesto musíme promyslet, jak větná forma v každém kroku vypadá. Potřebujeme v ní totiž najít taková podslova, která se při použití pravidla musí nebo nesmí vyskytovat. Často je nutné tvořit pravidla gramatiky tak, aby zaváděla různé pomocné neterminály. Vlastně si do větné formy poznačíme, jaké pravidlo bylo použito v tomto kroku. Na základě takových značek se vybírá pravidlo pro další krok. Podobným způsobem se navrhují i gramatiky s náhodným kontextem. Nemůžeme však využít rozhodování na základě podslov. Proto musíme pravidla a jejich povolené a zakázané kontexty promyslet tak, aby se mohla použitelná pravidla určit jen podle neterminálů ve větné formě. Obecně je návrh takové gramatiky složitější než návrh polopodmíněné.

Zdálo by se, že v uspořádaných gramatikách si můžeme přímo zajistit požadované pořadí pravidel v derivaci. Podle uspořádání by mohlo být vždy ze všech použitelných největší to, které si přejeme aplikovat. Většinou ale po několika krocích potřebujeme užít pravidlo, kterému v aplikaci zabrání uspořádání nutné pro předchozí kroky. Často se využívají pomocná pravidla, která zablokují derivaci, pokud mohou být použita. Můžeme si jimi vynutit požadovaný postup při generování terminálního slova. Nějakému pomocnému pravidlu dáme prioritu větší než všem, která by vedla ke slovům nepatřícím do jazyka. Tato pravidla tedy použít nemůžeme, protože pomocné je větší podle uspořádání. Pomocné by zase zablokovalo derivaci. Použitelná zůstanou jen pravidla vedoucí ke správnému terminálnímu slovu.

Není jednoduché zajistit, aby uspořádání neobsahovalo spory, aby byla použitelná v každém kroku jen pravidla, která chceme, a aby uspořádání nutné v jednom kroku nebránilo v aplikaci pravidla v dalším kroku.

Gramatiky s částečným paralelismem vyžadují trochu jiné myšlení při návrhu. Musíme vždy počítat s tím, že se pravidla aplikují současně. Jednotlivé typy paralelních gramatik jsou vhodné

jen pro určité jazyky. Dokáží je generovat snadněji, než sekvenčně řízené gramatiky. Pro jazyky s jinými vlastnostmi jsou však nevhodné. Gramatika pro ně se navrhuje těžko nebo vůbec neexistuje. Je to vidět na triviálních příkladech, které jsme si u jednotlivých typů uvedli. Trochu odlišná je k-jednoduchá maticová gramatika. Splnit všechny požadavky na její tvar není při návrhu snadné.

Ze všech uvedených typů řízených gramatik se nejpřirozeněji a nejsnadněji navrhují programované gramatiky. Z dalších jsou ještě z hlediska návrhu vhodné maticové a regulárně řízené gramatiky. Ze skupiny gramatik řízených kontextovými vlastnostmi jsou asi nejvhodnější polopodmíněné, ale v porovnání s gramatikami řízenými sekvencemi pravidel je jejich návrh složitější.

3.4 Shrnutí

Porovnali jsme řízené gramatiky z několika hledisek. Nyní výsledky shrneme a pokusíme se určit nejlepší gramatiku pro praktické využití.

Ve všech porovnáních se mezi nejlepšími umístily programované gramatiky. Z toho důvodů se zdají pro praxi nejpoužitelnější. Maticové gramatiky občas potřebují nějaká pomocná pravidla, ale jsou snadno implementovatelné a mají velkou generativní sílu. Z gramatik řízených kontextovými podmínkami se zdá nejvhodnější polopodmíněná. Její návrh není tak snadný jako návrh programované gramatiky. Simulace potřebuje více operací, protože musí v každém kroku hledat podslova. Je to však snazší, než kontrola celé větné formy v derivaci podle podmíněné gramatiky. Při implementaci není nutné vymýšlení žádných složitých algoritmů. Hledání podslova je běžné. Generativní síla polopodmíněných gramatik je dostatečně velká. V porovnání s ostatními podobně řízenými gramatikami jsou celkově nejlepší.

Z gramatik s částečným paralelismem má jen rozptýlená kontextová generativní sílu na úrovni obecných gramatik. Její implementace není náročná. Při návrhu ale vyžaduje trochu jiný způsob uvažování než sekvenční gramatiky.

4. Praktický příklad

Dosud uvedené příklady sice naznačily výhody a nevýhody řízených gramatik, ale pro praktické využití nám moc neukázaly. Proto si nyní uvedeme složitější příklad vycházející z programátorské praxe.

Pro jednoznačný popis programovacích jazyků se často využívají bezkontextové gramatiky. Některé vlastnosti těchto jazyků jsou kontextové. Jedná se hlavně o nutnost deklarace proměnných před jejich užitím. Tu bezkontextovými gramatikami nepopíšeme. Proto například při použití bezkontextové gramatiky v generátoru překladače vyhoví následné syntaktické analýze kód, který nesplňuje všechny požadavky. Deklarace se potom kontrolují tabulkami symbolů a jinými mechanizmy.

Abychom zajistili generování správného zdrojového kódu se všemi vlastnostmi danými specifikací jazyka, musíme použít gramatiku s větší generativní silou. Může to být klasická kontextová nebo obecná generativní gramatika. Výhodnější je ale použití některého typu řízené gramatiky. To si ukážeme na konkrétním jednoduchém programovacím jazyce. Popíšeme jej řízenou gramatikou a klasickou. Z řízených gramatik vyšla v porovnání v kapitole 3 nejlépe programovaná. Proto použijeme právě ji. Z Chomského hierarchie využijeme obecnou generativní gramatiku. Na popsání jazyka by asi stačila kontextová, ale obecná potřebuje méně pravidel, je přehlednější a konstruuje se snadněji.

4.1 Popis ukázkového jazyka

Program v ukázkovém jazyce je tvořen volitelnou sekvencí deklarací proměnných následovanou povinnou sekvencí příkazů. Všechny příkazy i deklarace jsou ukončeny středníkem.

4.1.1 Deklarace

Proměnné jsou dvou typů – `int` pro celá čísla a `boolean` pro logické hodnoty. Identifikátor tvoří řetězec znaků. Proměnné užité v sekci příkazů musí být deklarovány v sekci deklarací podle datového typu jedním z těchto příkazů:

```
int ijmeno;
boolean bjmeno;
```

Pro zjednodušení typové kontroly povinně začínají identifikátory celočíselných proměnných znakem ’i’ a logických proměnných znakem ’b’. Následuje alespoň jedno další písmeno. Nemusíme tak kontrolovat, jestli již proměnná stejného jména nebyla definována dříve s jiným datovým typem. Příkazů s deklarací proměnné může být libovolný počet. Některá proměnná může být deklarovaná i vícekrát, ale vždy se stejným datovým typem. Deklarovaná proměnná nemusí být použita v žádném příkazu.

4.1.2 Celočíselné výrazy

Celočíselné výrazy jsou tvořeny z celočíselných proměnných, konstant, závorek, aritmetických operátorů a volání funkce `readint()` bez parametrů pro načtení hodnoty ze vstupu. Operátory jsou unární i binární minus a binární $*$, $/$, $\%$, $+$ s významem násobení, celočíselné dělení, zbytek po celočíselném dělení, sčítání. Sčítání a odčítaní má menší prioritu než multiplikativní operátory, největší prioritu má unární mínus. Konstanty jsou běžná celá čísla v desítkové soustavě. Příklady:

```
iprom * (readint() + 10 - (-5))
5/4
readint()
```

Tyto výrazy jsou samostatně nepoužitelné. Musí být součásti logického výrazu nebo příkazu, které budou popsány níže.

4.1.3 Logické výrazy

Logické výrazy jsou tvořeny z logických proměnných, konstant `true` a `false`, volání funkce `readbool()` bez parametrů pro načtení hodnoty ze vstupu, závorek, logických operátorů a relačními operátory porovnaných celočíselných výrazů. Operátory jsou unární ! pro negaci, binární $\&\&$ pro logický součin a $||$ pro logický součet. Největší prioritu má !, nejmenší $||$. Relační operátory jsou $==, <, >, >=, <=, !=$ s významy jako v jazycích C a JAVA (pro gramatiku není význam důležitý). Relační operátory mají menší prioritu než všechny logické i aritmetické operátory. Příklady:

```
10 < 3
true || readbool()
(8+6 < iprom) || bprom
```

I tyto výrazy je nutno použít v níže popsaných příkazech. Samostatně nemají význam.

4.1.4 Příkazy

Jazyk obsahuje tři druhy příkazů. Nachází se v sekci příkazů, tzn. za všemi požadovanými deklaracemi. Prvním příkazem je přiřazení hodnoty proměnné. Logické proměnné lze přiřadit libovolný logický výraz, celočíselné proměnné celočíselný výraz. Např.

```
iprom = 10;
iprom2 = 328 + readint() * (iprom + 78);
bprom = iprom < 3;
bprom2 = bprom1 || bprom;
```

Další dva příkazy jsou podobné a slouží k vypsání hodnoty na výstup. `printint()` s celočíselným výrazem v závorkách vypíše číslo, `printbool()` s logickým výrazem v závorkách vypíše logickou hodnotu. Např.

```
printint(10+24);
printint(iprom);
printbool(bprom || (8<1));
```

4.2 Řešení programovanou gramatikou

Programovanou gramatiku jsme zvolili, protože patřila ve všech porovnáních řízených gramatik mezi nejlepší. Navíc nám poskytuje možnost rozdělení pravidel na podprogramy. Gramatika tak bude čitelnější a přehlednější. Obecný popis tvaru a funkce podprogramů byl uveden v části 3.3.2.

Pro upřesnění si uvedeme všechny neterminály a terminály, které se v podprogramech objeví.
$N = \{S, DECLS, STMTS, STMT, VAR, EXPR, BE, BEX, BT, BF, IE, IT, IF, BPR, IPR, DECL, IVAR, BVAR, NAME, CONT, RELOP, ADDOP, MULOP\}$
$T = \{printint, (,), ;, , printbool, readint, readbool, \&\&, ||, !, int, boolean, a, b, i, <, ==, +, -, *, /, \%\}$
Počáteční neterminál je S.

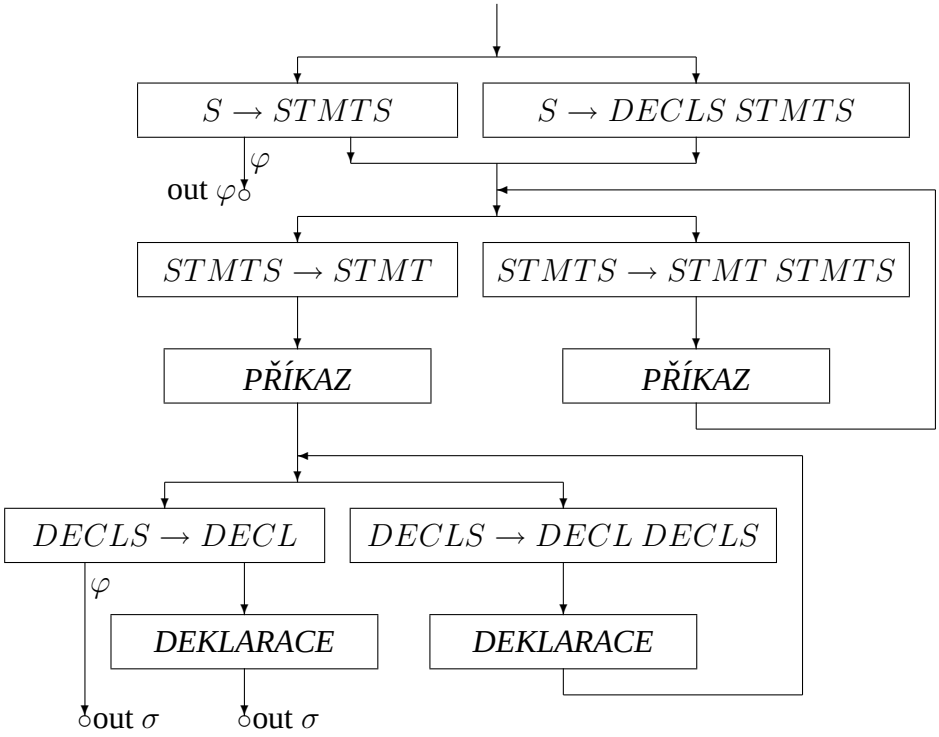
Tzv. bílými znaky (konec řádku, mezera, tabelátor apod.) se zabývat nebudeme. Jejich doplnění by ukázkovou gramatiku zbytečně znepřehlednilo.

4.2.1 Struktura zdrojového kódu

První podprogram jménem *STRUKTURA* zajišťuje vygenerování základní struktury zdrojového kódu. Jde vlastně o hlavní program, který přepíše startovací neterminál na terminální slovo patřící do jazyka. Využívá na to podprogramy pro vytvoření deklarace a příkazu uvedené níže. Pravidla jsou následující:

$\rightarrow a_1$: ($S \rightarrow STMTS$, $\{a_3, a_4\}$, {out φ})
$\rightarrow a_2$: ($S \rightarrow DECLS \quad STMTS$, $\{a_3, a_4\}$, ϕ)
a_3: ($STMTS \rightarrow STMT \quad STMTS$, $\{a_7\}$, ϕ)
a_4: ($STMTS \rightarrow STMT$, $\{a_8\}$, ϕ)
a_5: ($DECLS \rightarrow DECLS \quad DECL$, $\{a_9\}$, ϕ)
a_6: ($DECLS \rightarrow DECL$, $\{a_{10}\}$, {out σ})
a_7: (*PŘÍKAZ* , $\{a_3, a_4\}$, ϕ)
a_8: (*PŘÍKAZ* , $\{a_5, a_6\}$, ϕ)
a_9: (*DEKLARACE* , $\{a_5, a_6\}$, ϕ)
a_{10}: (*DEKLARACE* , {out σ} , ϕ)

Na obrázku 4.1 je přehledně zobrazeno, jaké pořadí pravidel při derivaci vynutí jednotlivá pole úspěchu a neúspěchu. Všechny neoznačené přechody mezi pravidly jsou po klasické aplikaci, přechody označené φ po použití v módu testování výskytu. Nejprve se vytvoří neterminál pro sekci příkazů a volitelně i pro sekci deklarací. Potom v cyklu vytváří neterminál pro jeden příkaz a ihned se pomocí volání podprogramu *PŘÍKAZ* popsaného v části 4.2.2 přepíše. Nevzniknou jen terminály, ale i neterminály $IVAR$ a $BVAR$. V posledním průchodu cyklem se neterminál $STMT$ vytvoří pravidlem a_4. I ten je přepsán podprogramem *PŘÍKAZ*. Po jeho



Obrázek 4.1: Pořadí pravidel při derivaci podle podprogramu *STRUKTURA*

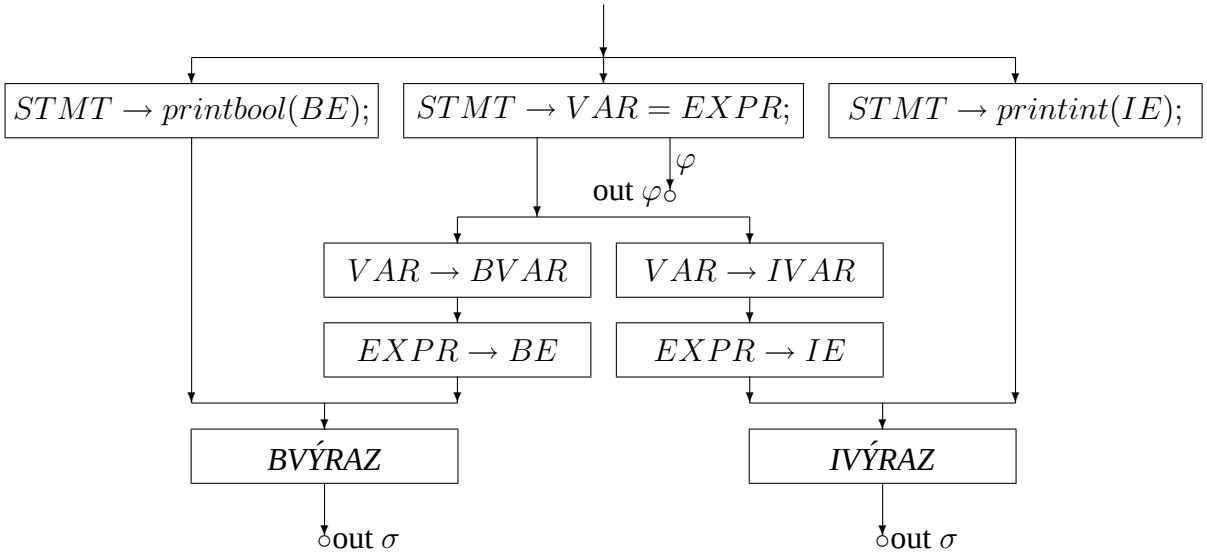
skončení potřebujeme pokračovat jinými pravidly. Využijeme tak jednu z výhod podprogramu. Každé použití může mít jiná pole úspěchu i neúspěchu. Stačí podprogramu přiřadit dvě různá označení. Kdybychom situaci chtěli řešit stejně bez podprogramu, museli bychom mít dvakrát s různými poli všechna pravidla, která náš podprogram tvoří.

Pokud nebyl zaveden neterminál $DECLS$, končí derivace po vytvoření všech příkazů aplikací pravidla a_6 ve smyslu testování výskytu. Jinak se pravidlem a_5 nebo a_6 vytvoří jeden neterminál $DECL$ a ihned je pomocí podprogramu popsaného v části 4.2.3 přepsán na deklaraci. Podprogram současně může přepsat některé výskyty neterminálů $BVAR$ nebo $IVAR$, které zůstaly v sekci příkazů. Po použití pravidla a_6, a přepsání vytvořeného $DECL$, musí vzniknout konečné terminální slovo (zdrojový kód vyhovující specifikaci programovacího jazyka). Derivace tím končí. Podprogram *DEKLARACE* potřebujeme dvakrát s odlišným označením ze stejného důvodu jako dříve diskutovaný *PŘÍKAZ*.

4.2.2 Příkazy

Druhý podprogram se jmenuje *PŘÍKAZ*. Slouží k přepisu jednoho neterminálu $STMT$ na slovo složené z neterminálů $IVAR$, $BVAR$ a terminálů. Obsahuje následující pravidla:

$\rightarrow b_1$: ($STMT \rightarrow VAR \quad = \quad EXPR \quad ; \quad , \{b_4, b_5\}, \{\text{out } \varphi\}$)
$\rightarrow b_2$: ($STMT \rightarrow printint \quad (\quad IE \quad) \quad ; \quad , \{b_8\} \quad , \phi \quad$)
$\rightarrow b_3$: ($STMT \rightarrow printbool \quad (\quad BE \quad) \quad ; , \{b_9\} \quad , \phi \quad$)
b_4: ($VAR \rightarrow BVAR \quad , \{b_6\} \quad , \phi \quad$)
b_5: ($VAR \rightarrow IVAR \quad , \{b_7\} \quad , \phi \quad$)
b_6: ($EXPR \rightarrow BE \quad , \{b_9\} \quad , \phi \quad$)
b_7: ($EXPR \rightarrow IE \quad , \{b_8\} \quad , \phi \quad$)
b_8: (*IVÝRAZ* $, \{\text{out } \sigma\}, \phi \quad$)
b_9: (*BVÝRAZ* $, \{\text{out } \sigma\}, \phi \quad$)



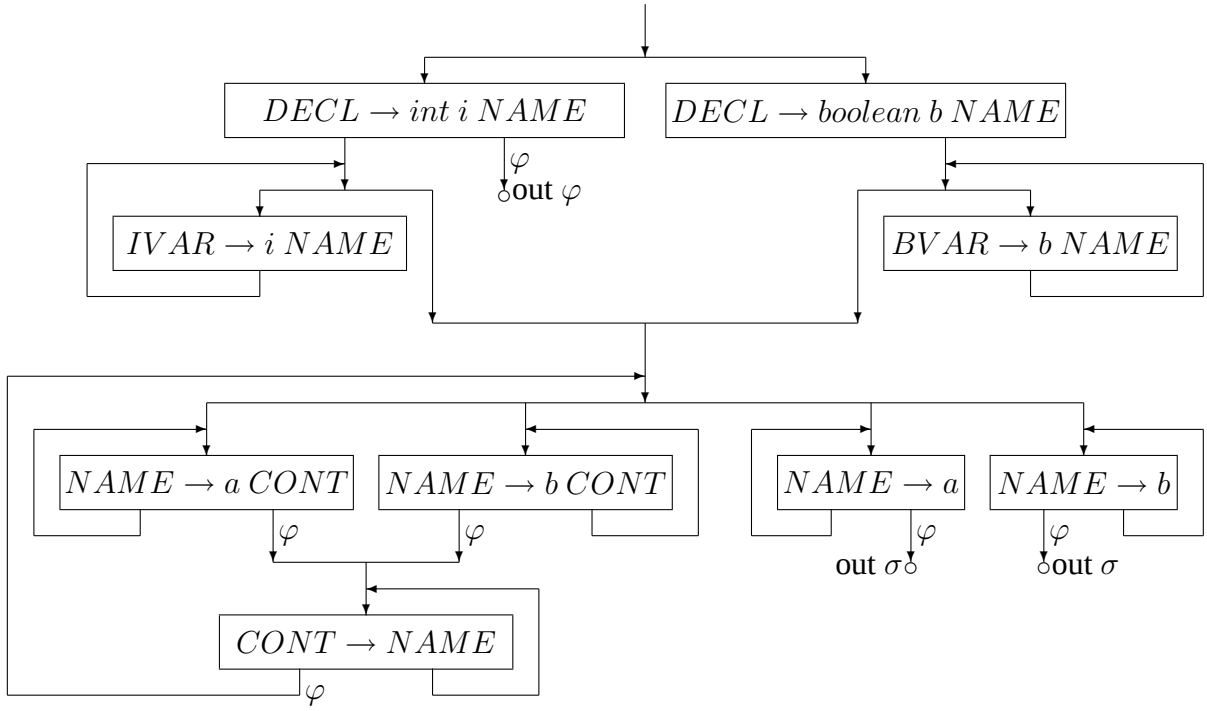
Obrázek 4.2: Pořadí pravidel při derivaci podle podprogramu *PŘÍKAZ*

Na obrázku 4.2 je znázorněno, jaké pořadí pravidel v podprogramu *PŘÍKAZ* je povoleno poli úspěchu a neúspěchu. Na začátku se volí jedna ze tří větví. Každá odpovídá jednomu druhu příkazu. Všechny větve končí voláním podprogramu. Podprogram tvořící celočíselný výraz je popsán v části 4.2.4 a logický výraz v 4.2.5. Neterminály $IVAR$ a $BVAR$ se přepíší až při tvorbě deklarací. Tím zajistíme, že každá použitá proměnná bude předem deklarována.

4.2.3 Deklarace

Další podprogram se jmenuje *DEKLARACE* a slouží k přepisu jednoho neterminálu $DECL$. Současně však může přepsat jeden nebo více výskytů $IVAR$ nebo $BVAR$.

$\rightarrow c_1$: $(DECL \rightarrow int \quad i \quad NAME \quad ; \quad , \{c_3, c_5\text{–}c_8\}, \{\text{out } \varphi\})$
$\rightarrow c_2$: $(DECL \rightarrow boolean \quad b \quad NAME \quad ;, \{c_4\text{–}c_8\} \quad , \phi \quad)$
c_3: $(IVAR \rightarrow i \quad NAME \quad , \{c_3, c_5\text{–}c_8\}, \phi \quad)$
c_4: $(BVAR \rightarrow b \quad NAME \quad , \{c_4\text{–}c_8\} \quad , \phi \quad)$
c_5: $(NAME \rightarrow a \quad CONT \quad , \{c_5\} \quad , \{c_9\} \quad)$
c_6: $(NAME \rightarrow b \quad CONT \quad , \{c_6\} \quad , \{c_9\} \quad)$
c_7: $(NAME \rightarrow a \quad , \{c_7\} \quad , \{\text{out } \sigma\})$
c_8: $(NAME \rightarrow b \quad , \{c_8\} \quad , \{\text{out } \sigma\})$
c_9: $(CONT \rightarrow NAME \quad , \{c_9\} \quad , \{c_5\text{–}c_8\})$



Obrázek 4.3: Pořadí pravidel při derivaci podle podprogramu *DEKLARACE*

Pořadí pravidel při derivaci podle podprogramu *DEKLARACE* je znázorněno na obrázku 4.3. Na začátku je potřeba zvolit, jestli se bude deklarovat celočíselná nebo logická proměnná. Potom se přepíše libovolný počet výskytů neterminálů $IVAR$, je-li zvolena deklarace celočíselné proměnné, nebo $BVAR$ při volbě logické proměnné. Vznikne nám několik výskytů neterminálu $NAME$. Tyto se potom budou přepisovat vždy stejně. Deklaraci proměnné tak bude odpovídat několik použití.

Pro přepsání $NAME$ můžeme zvolit jedno z pravidel c_5–c_8. Vybrané je potom použito na všechny výskyty $NAME$. Když se již ve větné formě žádné $NAME$ nevyskytuje, použijeme stejné pravidlo ve smyslu testování výskytu. Bylo-li zvoleno c_7 nebo c_8, podprogram končí. Po pravidlech c_5 a c_6 vznikne několik výskytů neterminálu $CONT$. Ty jsou přepsány na $NAME$ a pokračuje se znovu volbou c_5–c_8.

Uvedený podprogram generuje identifikátory jen z písmen a a b. Pro každé další písmeno by se musela přidat dvě pravidla. Jedno ve tvaru c_5 se sebou samým v poli úspěchu a c_9 v poli neúspěchu. Druhé přidané by mělo tvar c_7, v poli úspěchu sebe a $out\ \sigma$ v poli neúspěchu. Označení všech nových pravidel by se muselo přidat do množiny σ pravidel c_1–c_4 a do φ pravidla c_9.

4.2.4 Celočíselné výrazy

Podprogram na tvorbu celočíselných výrazů se jmenuje *IVÝRAZ*. Přepíše právě jeden výskyt neterminálu IE na terminály tvořící celočíselný výraz, který odpovídá specifikaci programovacího jazyka.

$\rightarrow d_1$: $(IE \rightarrow IT$, $\{d_3, d_4\}$, {out φ})
$\rightarrow d_2$: $(IE \rightarrow IE\ \ ADDOP\ \ IT$, $\{d_1, d_2\}$, ϕ)
d_3: $(IT \rightarrow IF$, $\{d_5$–$d_6\}$, ϕ)
d_4: $(IT \rightarrow IT\ \ MULOP\ \ IF$, $\{d_3, d_4\}$, ϕ)
d_5: $(IF \rightarrow IPR$, $\{d_7$–$d_{10}\}$, ϕ)
d_6: $(IF \rightarrow -\ \ IPR$, $\{d_7$–$d_{10}\}$, ϕ)
d_7: $(IPR \rightarrow IVAR$, $\{d_{13}$–$d_{15}\}$, ϕ)
d_8: $(IPR \rightarrow (\ \ IE\ \ )$, $\{d_1, d_2\}$, ϕ)
d_9: $(IPR \rightarrow readint\ \ (\ \ )$, $\{d_{13}$–$d_{15}\}$, ϕ)
d_{10}: $(IPR \rightarrow intconst$, $\{d_{13}$–$d_{15}\}$, ϕ)
d_{11}: $(ADDOP \rightarrow +$, $\{d_3, d_4\}$, {out σ})
d_{12}: $(ADDOP \rightarrow -$, $\{d_3, d_4\}$, ϕ)
d_{13}: $(MULOP \rightarrow *$, $\{d_5$–$d_6\}$, $\{d_{11}, d_{12}\}$)
d_{14}: $(MULOP \rightarrow /$, $\{d_5$–$d_6\}$, ϕ)
d_{15}: $(MULOP \rightarrow \%$, $\{d_5$–$d_6\}$, ϕ)

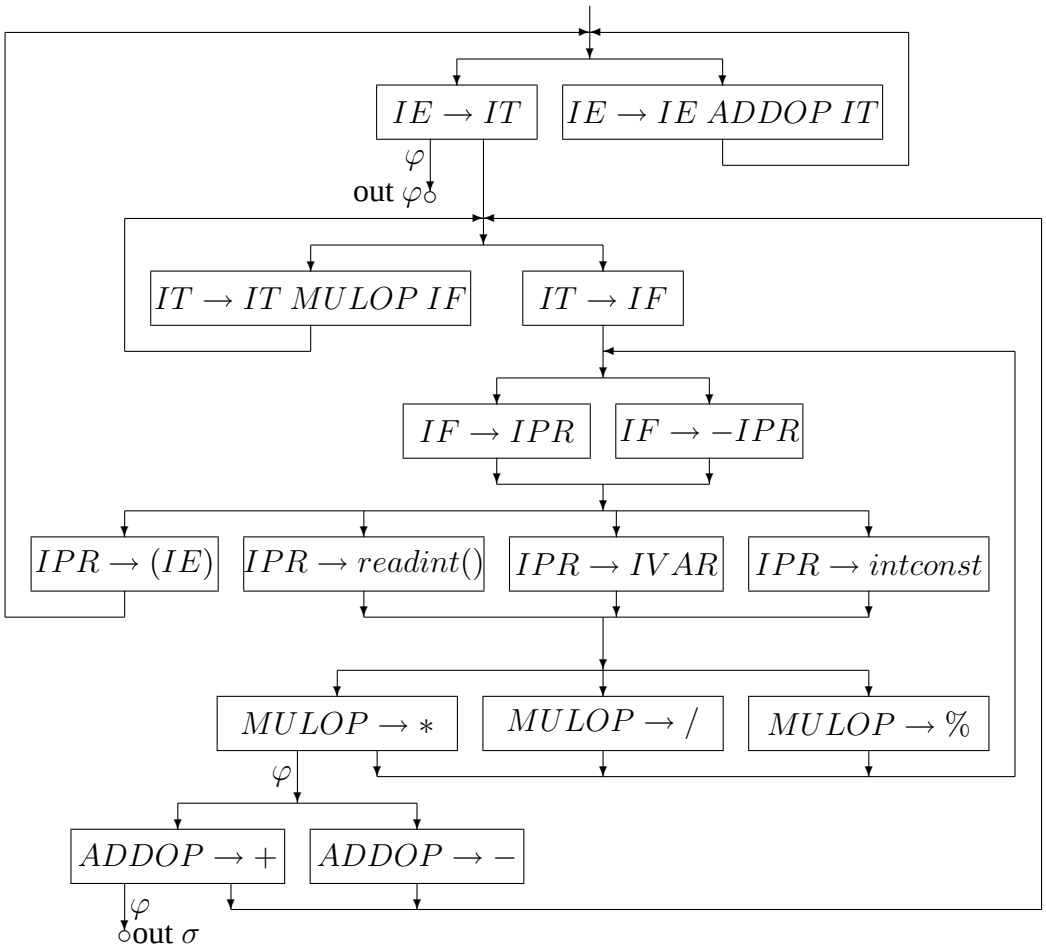
Pořadí pravidel vyplývající z obsahu polí úspěchu a neúspěchu je znázorněno na obrázku 4.4. Derivace začíná opakováním pravidla d_2 a následně užitím d_1. Vznikne forma tvaru

$$IT\ ADDOP\ IT\ ADDOP \ldots ADDOP\ IT.$$

Potom se aplikuje opět opakovaně pravidlo d_4 až do použití pravidla d_3. V té chvíli je jeden výskyt neterminálu IT nahrazen

$$IF\ MULOP\ IF\ MULOP \ldots MULOP\ IF.$$

Následně je vybrán jeden výskyt IF a přepsán na IPR. Přitom může vzniknout unární mínus. Na IPR se aplikuje některé z pravidel $d_7 - d_{10}$. Je-li to d_8, začne derivace znovu od d_2. Ostatní pravidla d_7, d_9 a d_{10} přepíší IPR na terminál. Potom se aplikuje pravidlo přepisující $MULOP$. Podaří-li se to klasickým způsobem, musí ve větné formě existovat ještě jeden IF tvořící druhý operand takto vytvořeného multiplikativního operátoru. Toto IF je následně přepsáno stejným způsobem (d_5 nebo d_6). Provede-li se aplikace pravidla d_{13} ve smyslu testování výskytu, nevyskytuje se ve větné formě IF. Tím byly získáno terminální slovo z jednoho výskytu IT. Potom se přepíše $ADDOP$. Povede-li se to klasicky, musí existovat ještě alespoň jeden IT tvořící



Obrázek 4.4: Pořadí pravidel při derivaci podle podprogramu *IVÝRAZ*

druhý operand k vytvořenému aditivnímu operátoru. Pokračuje se tedy opět přepisem IT od d_3 nebo d_4. Provede-li se aplikace pravidla d_{11} ve smyslu testování výskytu, nevyskytují se ve větné formě žádné neterminály. Než jsme testovali aditivní operátory, nebyl již ve větné formě žádný multiplikativní. Nejsou-li ve větné formě nepřepsané neterminály pro operátory, nemohou tam být ani neterminály pro operandy. Podprogram proto končí.

V popsaném podprogramu je generován terminál $intconst$, který zastupuje všechny možné celočíselné konstanty. Kdybychom je chtěli generovat správně, stačí změnit $intconst$ na neterminál a dodat pravidla jednoduchého regulárního jazyka pro tvorbu celého čísla.

4.2.5 Logické výrazy

I logické výrazy jsou tvořeny podprogramem. Jmenuje se *BVÝRAZ*. Přepíše právě jeden výskyt neterminálu BE na správně utvořený logický výraz.

$\rightarrow e_1$: ($BE \rightarrow IE \quad RELOP \quad IE$, $\{e_{16}\}$, ϕ)
$\rightarrow e_2$: ($BE \rightarrow BEX$, $\{e_3, e_4\}$, {out φ})
e_3: ($BEX \rightarrow BT$, $\{e_5, e_6\}$, ϕ)
e_4: ($BEX \rightarrow BEX \quad || \quad BT$, $\{e_3, e_4\}$, ϕ)
e_5: ($BT \rightarrow BF$, $\{e_7$–$e_8\}$, {out σ})
e_6: ($BT \rightarrow BT \quad \&\& \quad BF$, $\{e_5, e_6\}$, ϕ)
e_7: ($BF \rightarrow BPR$, $\{e_9$–$e_{13}\}$, $\{e_5, e_6\}$)
e_8: ($BF \rightarrow ! \quad BPR$, $\{e_9$–$e_{13}\}$, ϕ)
e_9: ($BPR \rightarrow BVAR$, $\{e_7$–$e_8\}$, ϕ)
e_{10}: ($BPR \rightarrow (\quad BE \quad)$, $\{e_1, e_2\}$, ϕ)
e_{11}: ($BPR \rightarrow readbool \quad (\quad)$, $\{e_7$–$e_8\}$, ϕ)
e_{12}: ($BPR \rightarrow true$, $\{e_7$–$e_8\}$, ϕ)
e_{13}: ($BPR \rightarrow false$, $\{e_7$–$e_8\}$, ϕ)
e_{14}: ($RELOP \rightarrow ==$, $\{e_{17}\}$, ϕ)
e_{15}: ($RELOP \rightarrow <$, $\{e_{17}\}$, ϕ)
e_{16}: (*IVÝRAZ* , $\{e_{14}, e_{15}\}$, ϕ)
e_{17}: (*IVÝRAZ* , $\{e_7, e_8\}$, ϕ)

Na obrázku 4.5 je znázorněno pořadí pravidel při derivaci podle podprogramu *BVÝRAZ*, které je opět dáno tvarem polí úspěchu a neúspěchu. Pravidla e_{12} a e_{13} jsou téměř stejná, proto jsou na obrázku znázorněna společně jako $BF \rightarrow const$. Logické výrazy se dělí na dva druhy. Jim odpovídají dvě větve derivace. Jedna tvoří porovnání dvou celočíselných výrazů relačním operátorem a druhá skládá výraz z logických hodnot a operátorů logického sčítání a násobení.

Začne-li derivace aplikací pravidla e_1, vytvoří se následně jeden celočíselný výraz, relační operátor a nakonec druhý celočíselný výraz. Oba celočíselné výrazy vzniknou po aplikaci podprogramu, který je popsán v 4.2.4. Potom se ještě překontroluje aplikací pravidel e_7 a e_5 v módu testování výskytu, jestli se nenacházejí ve větné formě neterminály BF a BT. Je to nutné, protože relačními operátory vytvořený výraz může být součástí většího, který je tvořen logickými operátory. V takovém případě by se mohly ve větné formě nacházet nepřepsané operandy logických operátorů – neterminály BF a BT.

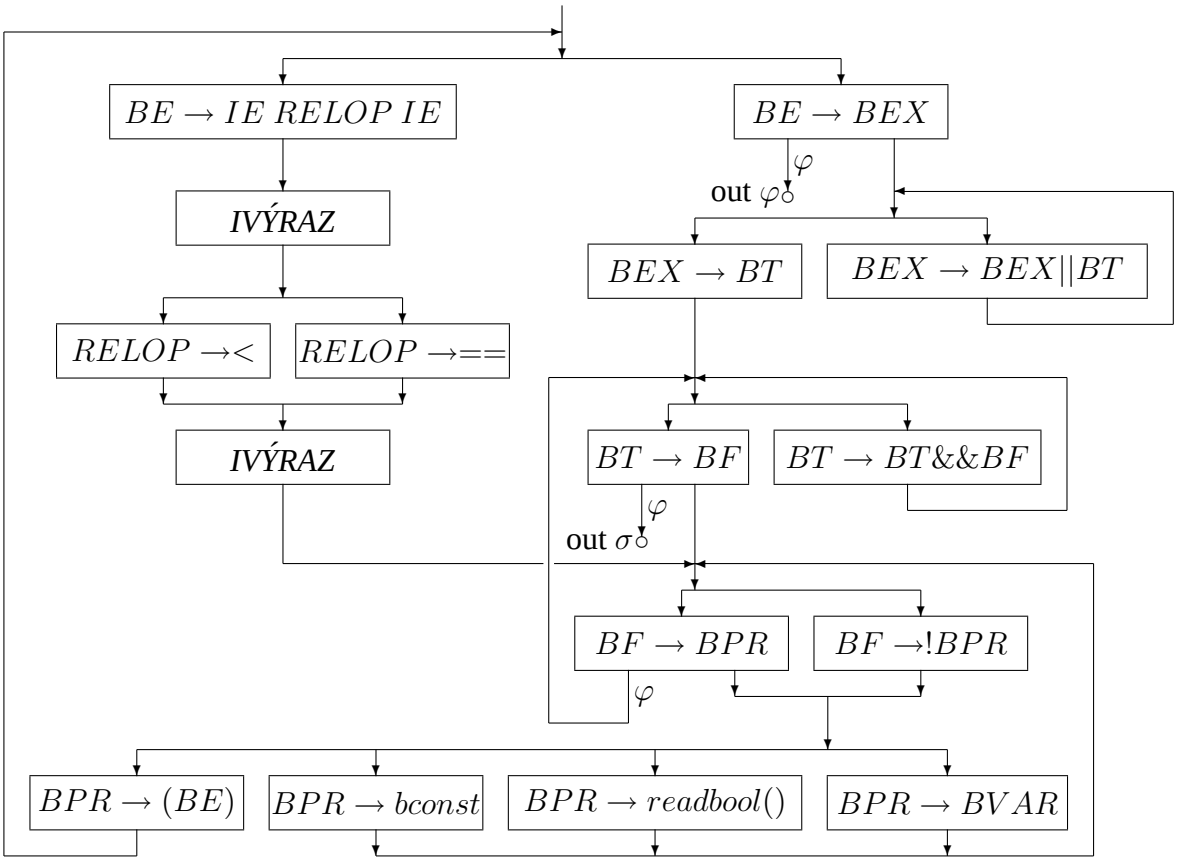
Na začátku druhé větve se aplikuje několikrát e_4 a potom jednou e_3. Vznikne větná forma tvaru

$$BT \,||BT \,|| \ldots || \,BT$$

Na ni můžeme několikrát použít pravidlo e_6 a po e_5 je jeden výskyt BT přepsán na

$$BF \,\&\&BF \,\&\& \ldots \&\& \,BF$$

Na jeden z neterminálů BF aplikujeme e_7 nebo e_8. Tím vygenerujeme případný unární operátor negace. Na vzniklý BPR můžeme použít e_9–e_{13}. Po e_{10} se začne znovu od pravidla e_1 nebo e_2. Ostatní pravidla přepíší BPR na terminální slovo. Pokračuje se znovu výběrem z pravidel e_7 a e_8. Po přepsání všech BF se použije e_7 ve smyslu testování výskytu. Ve větné formě se ještě mohou nacházet BT. Proto se znovu pokračuje pravidly e_5 a e_6. Když jsou přepsána i všechna BT, aplikuje se pravidlo e_5 v módu testování výskytu a podprogram končí.



Obrázek 4.5: Pořadí pravidel při derivaci podle podprogramu *BVÝRAZ*

V podprogramu jsou uvažovány jen dva relační operátory. Pro každý další by stačilo přidat pravidlo podobné e_{14}. Označení nově přidaného pravidla by muselo být doplněno do pole úspěchu pravidla e_{16}.

4.3 Řešení neřízenou (klasickou) gramatikou

Pro řešení klasickým způsobem je obecná generativní gramatika vhodnější než kontextová. Neklade totiž téměř žádná omezení na tvar pravidel. Můžeme tak přepisovat více neterminálů současně, nebo používat pravidla typu $AX \rightarrow XA$. Taková pravidla bychom mohli nahradit několika kontextovými, ale najít v nových pravidlech původní myšlenku prohození pořadí neterminálů by bylo pro čtenáře obtížnější.

Všechny řetězce velkých písmen v pravidlech jsou neterminály. Terminály jsou stejné jako v řešení programovanou gramatikou. Jejich množina byla uvedena v části 4.2. Množinu pravidel tvoří všechna, která si postupně uvedeme.

Začátek derivace může být stejný jako v programované gramatice. Vygenerujeme neterminál pro příkazy a volitelně pro deklarace. Ze $STMTS$ se vytvoří potřebný počet neterminálů pro

jednotlivé příkazy.

$S \rightarrow STMTS$ $\quad$ $S \rightarrow DECLS \quad STMTS$
$STMTS \rightarrow STMT$ $\quad$ $STMTS \rightarrow STMT \quad STMTS$

Pravidla pro tvorbu příkazů se od řešení programovanou gramatikou trochu liší. Pro každý datový typ máme jedno pravidlo tvořící příkaz přiřazení. V programované se nejprve vytvořilo netypové přiřazení a následně se rozhodlo o datovém typu.

$STMT \rightarrow printint\ (\ IE\ )\ ;$ $\quad$ $STMT \rightarrow printbool\ (\ BE\ )\ ;$
$STMT \rightarrow IVAR\ =\ IE\ ;$ $\quad$ $STMT \rightarrow BVAR\ =\ BE\ ;$

Pravidla pro tvorbu logických výrazů i celočíselných výrazů byla v programované gramatice navržena tak, že generují stejný jazyk i bez řízení. Neuvažujeme-li levou derivaci, probíhá výběr pravidel méně deterministicky než v programované gramatice. Můžeme je používat kdykoliv, kdy je to možné podle klasické definice aplikace. Determinističtější derivace podle řízené gramatiky je ale nevýhodou při návrhu. Musíme její průběh důkladně promyslet, aby mohly být vytvořeny všechny výrazy odpovídající specifikaci jazyka a aby se díky řízení derivace někde nezablokovala. Pro úplnost si pravidla uvedeme.

$IE \rightarrow IT$	$IE \rightarrow IE\ ADDOP\ IT$	$IT \rightarrow IF$
$IT \rightarrow IT\ MULOP\ IF$	$IF \rightarrow -\ IPR$	$IF \rightarrow IPR$
$IPR \rightarrow IVAR$	$IPR \rightarrow (\ IE\ )$	$IPR \rightarrow readint\ (\ )$
$IPR \rightarrow intconst$	$ADDOP \rightarrow +$	$ADDOP \rightarrow -$
$MULOP \rightarrow *$	$MULOP \rightarrow /$	$MULOP \rightarrow \%$
$BE \rightarrow IE\ RELOP\ IE$	$BE \rightarrow BEX$	$BEX \rightarrow BT$
$BEX \rightarrow BEX\ \|\|\ BT$	$BT \rightarrow BF$	$BT \rightarrow BT\ \&\&\ BF$
$BF \rightarrow !\ BPR$	$BF \rightarrow BPR$	$BPR \rightarrow BVAR$
$BPR \rightarrow (\ BE\ )$	$BPR \rightarrow readbool\ (\ )$	$BPR \rightarrow true$
$BPR \rightarrow false$	$RELOP \rightarrow ==$	$RELOP \rightarrow <$

Nejvíce se od programované gramatiky liší řešení deklarací. Jde o kontextovou vlastnost, která řízení potřebovala. Nyní budeme muset používat kontextová nebo obecná pravidla.

Pomocí několika pravidel si zajistíme, že nebude možné současně tvořit deklarace více proměnných. Neterminál $DECL$ pro deklaraci může vzniknout, když jsou W a DS vedle sebe. W se posouvá větnou formou doprava jen přes terminály. Než je jeden $DECL$ přepsán na terminální slovo, nemůže se W dostat k DS a jiný $DECL$ nevznikne.

$DECLS \rightarrow W\ DS$	$W\ DS \rightarrow W\ DECL\ DS$	$W\ DS \rightarrow DECL$
$W\ int \rightarrow int\ W$	$W\ boolean \rightarrow boolean\ W$	$W\ a \rightarrow a\ W$
$W\ b \rightarrow b\ W$	$W\ i \rightarrow i\ W$	$W\ ;\rightarrow;\ W$

Při tvorbě konkrétní deklarace si zavedeme kromě neterminálu $NAME$ pro tvorbu identifikátoru ještě dva pomocné. X bude sloužit jako levá zarážka. Y, resp. V, určuje, že se jedná o deklaraci celočíselné, resp. logické proměnné. Y i V putují větnou formou doprava přes libovolné terminály a neterminály $IVAR$, $BVAR$ a DS. Jiné neterminály se již ve větné formě

nemusí vyskytovat. Vždy je můžeme přepsat dříve, než začneme generovat deklarace. Nachází-li se Y vedle $IVAR$, může se $IVAR$ nahradit neterminálem $NAME$. Znamená to, že na daném místě se bude vytvářet stejný identifikátor, jaký právě deklarujeme. Půjde o použití deklarované proměnné. Poslední výskyt stejné proměnné se vytvoří jiným pravidlem. Vygeneruje se kromě $NAME$ ještě pravá zarážka Z a Y se změní na D. Obdobně se přepisují logické proměnné.

D vždy bude signalizovat, že všechny generované výskyty stejného identifikátoru jsou zrovna ve stejném tvaru. Na začátku je to pouze neterminál $NAME$, později i dosud vygenerovaná část identifikátoru. D vždy vznikne u pravé zarážky a putuje větnou formou až k levé. Když je D vedle X, vybere se pravidlo podle toho, jestli se má do identifikátorů přidat a nebo b. Neterminál A znamená přidání a, B přidání b. A i B putují větnou formou doprava až k pravé zarážce. Cestou se ke každému výskytu $NAME$ přidá požadovaný terminál. Nachází-li se A nebo B vedle pravé zarážky Z, rozhoduje se, jestli již jsou identifikátory hotové, nebo se budou tvořit dál. Nejsou-li dokončené, A nebo B se změní na D. To opět putuje doleva a cyklus se opakuje.

Pokud je identifikátor hotov, změní se A nebo B na C a odstraní se pravá zarážka. C také putuje až k levé zarážce. Cestou jsou odstraněny všechny výskyty neterminálu $NAME$. Když C dorazí až k levé zarážce X, jsou oba tyto neterminály vypuštěny. Deklarace jedné proměnné je vytvořena. Nešlo-li o poslední deklaraci, jsou mezi W a DS jen terminály. W se proto může posunout až k DS, a tím umožní generování další deklarace.

Pomocí dvou pravidel zajistíme deklaraci proměnné, která nebude použita. Vytvoříme jimi současně levou i pravou zarážku a identifikátor se generuje jen na jednom místě.

Pokud se v některém níže uvedeném pravidle vyskytuje $term$, znamená to, že pravidlo toho tvaru existuje pro všechny terminály kromě int, $boolean$ a i. V každém takovém pravidle jsou všechny výskyty $term$ nahrazeny stejným terminálem.

$DECLS \rightarrow int\ i\ X \quad NAME\ ;\ Y$
$DECLS \rightarrow boolean\ b\ X \quad NAME\ ;\ V$
$DECLS \rightarrow int\ i\ X \quad D \quad NAME\ ;\ Z$
$DECLS \rightarrow boolean\ b\ X \quad D \quad NAME\ ;\ Z$
$X \quad D \rightarrow X \quad A$
$Y \quad term \rightarrow term \quad Y$
$Y \quad BVAR \rightarrow BVAR \quad Y$
$Y \quad IVAR \rightarrow IVAR \quad Y$
$Y \quad IVAR \rightarrow NAME \quad Y$
$Y \quad IVAR \rightarrow D \quad NAME \quad Z$
$Y \quad DS \rightarrow DS \quad Y$
$A \quad NAME \rightarrow a \quad NAME \quad A$
$A \quad term \rightarrow term \quad A$
$A \quad IVAR \rightarrow IVAR \quad A$
$A \quad BVAR \rightarrow BVAR \quad A$
$A \quad DS \rightarrow DS \quad A$

$term \quad D \rightarrow D \quad term$
$NAME \quad D \rightarrow D \quad NAME$
$IVAR \quad D \rightarrow D \quad IVAR$
$BVAR \quad D \rightarrow D \quad BVAR$
$X \quad D \rightarrow X \quad B$
$V \quad term \rightarrow term \quad V$
$V \quad BVAR \rightarrow BVAR \quad V$
$V \quad IVAR \rightarrow IVAR \quad V$
$V \quad BVAR \rightarrow NAME \quad V$
$V \quad BVAR \rightarrow D \quad NAME \quad Z$
$V \quad DS \rightarrow DS \quad V$
$B \quad NAME \rightarrow b \quad NAME \quad B$
$B \quad term \rightarrow term \quad B$
$B \quad IVAR \rightarrow IVAR \quad B$
$B \quad BVAR \rightarrow BVAR \quad B$
$B \quad DS \rightarrow DS \quad B$

$A \quad Z \to D \quad Z$ $\qquad$ $B \quad Z \to D \quad Z$

$A \quad Z \to C$ $\qquad$ $B \quad Z \to C$

$term \quad C \to C \quad term$ $\qquad$ $NAME \quad C \to C$

$IVAR \quad C \to C \quad IVAR$ $\qquad$ $BVAR \quad C \to C \quad BVAR$

$DS \quad C \to C \quad DS$ $\qquad$ $X \quad C \to \lambda$

$DS \quad D \to D \quad DS$

Kdybychom chtěli generovat v identifikátorech i další znaky, museli bychom přidat pro každý z nich jeden nový neterminál s významem A a pravidla obdobná všem, ve kterých se vyskytuje A nebo a.

Řešení programovanou gramatikou je mnohem přirozenější. Nepotřebujeme v ní vymýšlet složité mechanismy jako putování neterminálů větnou formou. Kromě přirozenějšího návrhu má programovaná gramatika mnohem méně pravidel a tím je přehlednější a srozumitelnější pro čtenáře. Také derivace podle uvedené programované gramatiky proběhne v méně krocích a s méně nedeterminismy.

5. Závěr

Práce seznamuje čtenáře se základem teorie řízených gramatik. Neklade si za cíl nalezení nových druhů řízení. Definice gramatik jsou přebrány z literatury. Účelem práce je vysvětlit jejich princip srozumitelně. K tomu slouží i podrobně popsané příklady, které čtenáři pomohou pochopit mechanizmus jednotlivých typů řízení.

Řízené gramatiky jsme porovnali mezi sebou i s klasickými gramatikami z hlediska generativní síly, snadnosti implementace algoritmu derivace a přirozenosti návrhu. Tím čtenář získá představu o tom, která gramatika je k používání nejvhodnější. Z porovnání vychází nejlépe programovaná gramatika. Ta je použita k popisu jednoduchého programovacího jazyka. Abychom ukázali výhody řízení, je pro stejný jazyk uvedena klasická gramatika. Návrh programované gramatiky byl jednodušší a pro čtenáře je snadněji pochopitelný způsob, jakým zajišťuje generování požadovaného jazyka. Proto je programovaná gramatika vhodná pro praktické využití. I některé další řízené gramatiky by se mohly používat, jiné jsou ale z různých důvodů nevhodné. Konkrétní způsob, jakým by se řízené gramatiky v praxi využily, je možným námětem pro další práce.

Při dalším výzkumu by bylo vhodné dokázat, jestli leftmost omezení změní generativní sílu řízených gramatik. Také možnost použití všech pravidel ve smyslu testování výskytu by přinesla výhody při návrhu gramatiky i tvorbě algoritmu derivace, pokud by bylo dokázáno, že nebude snížena generativní schopnost.

Součástí práce je aplikace, která simuluje derivaci podle programované gramatiky. Program čtenáři umožní vyzkoušet si prakticky, jak probíhá řízená derivace pro jednoduché jazyky. Objektový návrh programu je vytvořen tak, aby umožnil snadné rozšíření pro další typy řízených i neřízených gramatik.

Literatura

[HFL–96] Dassow, J., Păun, Ch., Salomaa, A., Grammars with Controlled Derivations, *Handbook of Formal Languages Volume 2 (Linear Modeling: Background and Application)*, Springer-Verlang, Berlin-Heidelberg, 1997, 101–154

[Sal–73] Salomaa, A., *Formal Languages*, Academic Press, New York, San Francisco, London, 1973

Literatura uvedená v [HFL–96]

[IAT–79] Hopcroft, J. E., Ullman, J. D., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, 1979

[Sky–74] Skyum, S., Parallel context-free languages, *Inform. Control*, 26 (1974), 280–285

[Kyb–85] Dassow, J., Păun, Ch., Further remars on the complexity of regulated rewriting, *Kybernetika*, 21 (1985), 213–227

[Bul–89] Hinz, F., Dassow, J., An undecidability result for regular languages and its application to regulated rewriting, *Bulletin EATCS*, 38 (1989), 168–173

[Bul–95] Meduna, A., A trivial method of characterizing the family of recursively enumerable languages by scattered context grammars, *Bulletin EATCS*, 56 (1995), 104–106

[Pău–80] Păun, Ch., A new generative device: valence grammars, *Rev. Roum. Math. Pures Appl.*, 25 (1980), 911–924

[Sti–96] Stiebe, R., *Kantengrammatiken*, Dissertation, Magdeburg Univ., 1996

[Act–89] Gonczarowski, J., Warmuth, M. K., Scattered and context-sensitive rewriting, *Acta Informatica*, 20 (1989), 391–411

[Act–95] Meduna, A., Syntactic complexity of scattered context grammars, *Acta Informatica*, 32 (1995), 285–298

Seznam příloh

Obsah CD

simulator – program Simulátor programované gramatiky

- **src** – zdrojové kódy v jazyce JAVA
- **doc** – dokumentace vygenerovaná programem javadoc
- **classes** – přeložené soubory tříd (byte-code)
- **examples** – příklady několika gramatik ve formátu, který umí program přečíst
- **GrammarSimulator.jar** – přeložené soubory tříd (byte-code) v jar archívu
- **GrammarSimulator.htm** – html stránka pro spuštění programu ve formě appletu
- **run, run.bat** – dávkové soubory pro spuštění programu

text – text diplomové práce

- **src** – zdrojové kódy pro LaTeX a použité obrázky
- **grammars.pdf, grammars.dvi, grammars.ps** – text teoretické části diplomové práce v několika formátech
- **document.pdf** – text programátorské a uživatelské příručky