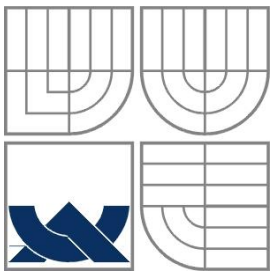
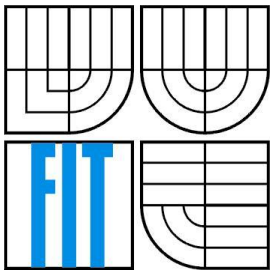


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# NEKLASICKÁ TÉMATA VÝZKUMU GRAMATIK S ROZPTÝLENÝM KONTEXTEM

TJD 2009/2010

AUTOR PRÁCE Ing. Peter Solár

BRNO 2010

# Obsah

# 1 Úvod

Tato práce, věnovaná především různým modifikacím gramatik s rozptýleným kontextem využitelným při překladu nejen programovacích jazyků je rozdělena do tří základních částí.

V první části budou připomenuty základy formálních jazyků, budou připomenuty teoretické modely, jako jsou gramatiky a automaty. Samozřejmě nebudou chybět základní informace o překladu programovacích jazyků.

Druhá část se bude týkat teorie o gramatikách s rozptýleným kontextem. Budou prezentovány různé modifikace, které mají vliv na jejich vyjadřovací schopnost

Poslední část se bude věnována modifikacím uvedených gramatik s ohledem na využití v překladu jazyků – nejprve půjde o generátory a samozřejmě nebude chybět překladová varianta gramatik s rozptýleným kontextem.

Tato práce čerpá především z knihy *Scattered Context Grammars and Their Applications*, jejímiž autory jsou prof. A. Meduna a J. Techet ([7]).

# 2 Teoretický základ

V této kapitole bych rád připomněl základní teoretický základ potřebný k lepšímu pochopení dalších částí této práce.

## 2.1 Základní pojmy

Nejprve se podívejme na základní matematické symboly a termíny.

*Abeceda* je neprázdná konečná množina prvků, nazývaných *symboly*. Nechť Σ je abeceda. Potom $\varepsilon$ je *prázdný řetězec* nad abecedou Σ (jedná se o řetězec délky 0). Pokud by $x$ byl řetězec nad abecedou Σ a $a$ je symbol $a \in \Sigma$, pak $xa$ je také řetězec nad abecedou Σ. *Konkatenace* dvou řetězců je jejich spojení za vzniku nového řetězce. Příkladem může být výše uvedený řetězec $xa$ (pozn. symbol $a$ lze chápat také jako řetězec délky 1). Výsledkem konkatenace jakéhokoliv řetězce s prázdným řetězcem je původní řetězec ($x\varepsilon = \varepsilon x = x$).

Ze symbolů abecedy Σ můžeme konkatenací vytvořit množinu neprázdných řetězců $\Sigma^+$. Když k této množině přidáme i prázdný řetězec, získáme množinu $\Sigma^*$. Tedy platí $\Sigma^* = \Sigma^+ + \{\varepsilon\}$.

Pro jakýkoliv řetězec $x \in \Sigma^*$ zavedeme $|x|$ označující jeho délku (počet všech symbolů, které se v daném řetězci nacházejí) a funkci $alph(w)$ označující množinu symbolů vyskytujících se v řetězci $w$.

## 2.2 Jazyky

Jazyk je z formálního hlediska podmnožina množiny všech možných řetězců nad danou abecedou. Máme-li abecedu Σ a jazyk $L$, můžeme tuto skutečnost zapsat jako $L \subseteq \Sigma^*$. Jazyky zpravidla dělíme na konečné a nekonečné – podle toho, zda obsahují konečný počet řetězců či nikoliv. Příkladem nekonečného jazyka může být jazyk $L = \{a^n b^n | n \geq 1\}$ nad abecedou $\Sigma = \{a, b\}$. Mezi konečné jazyky patří i $L = \emptyset$ s kardinalitou 0 (jazyk neobsahuje žádný řetězec) a jazyk $L = \{\varepsilon\}$ s kardinalitou 1 (jeden prázdný řetězec).

Mějme dva jazyky $L_1$ a $L_2$ nad abecedou $T$. Pro tyto jazyky můžeme definovat operace:

$$L_1 \cup L_2 = \{x: x \in L_1 \text{ nebo } x \in L_2\}$$

$$L_1 \cap L_2 = \{x: x \in L_1 \text{ a zároveň } x \in L_2\}$$

$$L_1 - L_2 = \{x: x \in L_1 \text{ a zároveň } x \notin L_2\}$$

$$L_1.L_2 = \{xy: x \in L_1, y \in L_2\}$$

$$L_1/L_2 = \{y: yx \in L_1 \text{ pro nějaké } x \in L_2\}$$

$$L_2 \backslash L_1 = \{y: xy \in L_1 \text{ pro nějaké } x \in L_2\}$$

$$L_1 \,//\, L_2 = \{x: x \in L_1/L_2 \text{ a žádný řetězec z } L_1/L_2 \text{ není prefixem } x\}$$

$$L_2 \,\backslash\backslash L_1 = \{x: x \in L_2 \backslash L_1 \text{ a žádný řetězec z } L_1 \backslash L_2 \text{ není prefixem } x\}$$

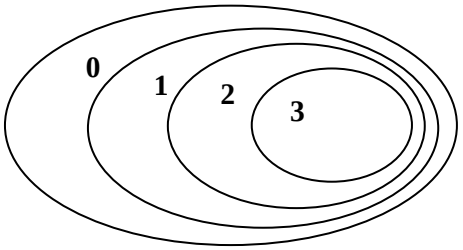
Jazyky můžeme popisovat dvěma základními modely. Prvním modelem jsou gramatiky. Ty podle svých gramatických pravidel generují (derivují) řetězce daného jazyka. Druhým modelem jsou automaty. Na rozdíl od gramatik však řetězce nevytvářejí, ale na základě svých pravidel rozhodují, zda vstupní řetězec patří do jazyka popisovaného tímto automatem.

### 2.2.1 Chomského klasifikace jazyků

Noam Chomsky v roce 1956 zavedl hierarchii formálních gramatik, generujících rekurzivně spočetné formální jazyky. Tyto gramatiky rozdělil do 4 základních typů.

| typ | gramatika | automat | jazyk |
|---|---|---|---|
| **0** | neomezená | Turingův stroj (TS) | rekurzivně vyčíslitelný (RE) |
| **1** | kontextová (CSG) | lineárně ohraničený TS | kontextový (CS) |
| **2** | bezkontextová (CFG) | zásobníkový automat (PDA) | bezkontextový (CF) |
| **3** | regulární | konečný automat (FA) | regulární |

Každý typ této klasifikace je přímou nadmnožinou dalšího typu (viz. Obrázek 1: Chomského hierarchie).




**Obrázek 1: Chomského hierarchie**

## 2.3 Gramatiky

Gramatiky jsou jedním ze základních stavebních prvků teorie formálních jazyků. Jádrem každé gramatiky jsou tzv. gramatické pravidla, jejichž pomocí dochází ke generování řetězců patřících do jazyka, který tato gramatika popisuje. V minulosti bylo u gramatik, na rozdíl od automatů, zavedeno velké množství modifikací. Proto uvedu jen ty základní, které jsou pro tuto práci zajímavé.

### 2.3.1 Bezkontextové gramatiky

Bezkontextové gramatiky jsou schopny generovat bezkontextové jazyky, tj. typu 2 podle Chomského klasifikace. Do této skupiny jazyků lze zařadit například i programovací jazyky. Bezkontextové gramatiky jsou v porovnání s regulárními gramatikami výrazně silnější. Mezi typické příklady jazyků generovaných bezkontextovými gramatikami patří $L(G) = \{a^n b^n | n \geq 1\}$.

**Definice 1:** *Bezkontextová gramatika* je čtveřice

$$G = (V, T, P, S),$$

kde

$V$ je úplná abeceda

$N$ je konečná množina neterminálních symbolů $N \subset V$

$T$ je konečná množina terminálních symbolů, $T = V - N$

$S \in N$ je počáteční neterminál

$P$ je konečná množina gramatických pravidel tvaru $A \to x,\ A \in N,\ x \in V^*$

# 3 Překladače

V současné době téměř každý programátor programuje v některém z vyšších programovacích jazyků. Těmto jazykům však procesor počítače nerozumí a proto je potřeba transformovat programy napsané programátorem do strojového jazyka. Při této transformaci samozřejmě požadujeme, aby program ve strojovém jazyce prováděl přesně to, co původní verze napsaná programátorem. Tato transformace se nazývá *překlad* a má ji na starosti program zvaný *překladač*.

*Překladač* je program, který na svém vstupu přijímá zdrojový program (ve zdrojovém jazyce) a na výstupu produkuje cílový program (v cílovém jazyce).

*Kompilátor* je speciální druh překladače, překládající programy z vyšších programovacích jazyků na ekvivalentní programy v nižších programovacích jazycích.

*Assembler* je program, překládající program ze strojového jazyka na ekvivalentní strojový kód.

V praxi se také můžeme setkat i s pojmy *dekompilátor* a *disassembler*. Jejich činnost je opačná k výše uvedeným pojmům.

*Interpret* je program, který je schopen přímo vykonávat program napsaný ve zdrojovém jazyce.

Překlad se obvykle skládá z následujících fází:

- lexikální analýza
- syntaktická analýza
- sémantická analýza
- generování vnitřní formy programu
- optimalizace
- generování cílového programu

Podle jednotlivých fází se nazývají i jednotlivé části kompilátoru.

*Lexikální analyzátor* je první částí kompilátoru. Jeho úkolem je projít kód zdrojového programu a rozdělit jej na posloupnost lexikálních symbolů a jednotlivým symbolům přiřadit typ (identifikátor, klíčové slovo, konstanta, …). Výstup lexikální analýzy se předá *syntaktickému analyzátoru*. Ten má zjistit, zda je zdrojový program napsán syntakticky správně (tedy uvedená posloupnost lexikálních symbolů patří do daného jazyka). Výstupem syntaktického analyzátoru je

obvykle *derivační strom*. *Sémantický analyzátor* kontroluje celou řadu dalších, tzv. sémantických aspektů (jsou proměnné deklarovány?, jsou proměnné správného typu?, apod.).

*Generátor kódu* vytvoří na základě syntaktické struktury cílový program. Součástí tohoto generování mohou být provedeny různé *optimalizace*.

V praxi se však první 3 fáze (lexikální, syntaktická a sémantická analýza) zpravidla prolínají a jednotlivé části kompilátoru spolupracují.

# 3.1 Formální překlad

Mějme abecedy $\Sigma_I$, $\Sigma_O$ a jazyky $L_I \subseteq \Sigma_I^*$, $L_O \subseteq \Sigma_O^*$. Překladem z jazyka $L_I$ do jazyka $L_0$ nazveme relaci

$$P \subseteq L_I \times L_O$$

$\Sigma_I$ je vstupní abeceda

$\Sigma_O$ je výstupní abeceda

Jestliže $(x, z) \in P$, pak slovo $z$ nazveme překladem věty $x$.

Překlad je relací, tedy může existovat více výstupů pro jednu konkrétní vstupní větu. To však v praxi může vést k problémům. Protože je překlad tvořen nekonečnou množinou dvojic slov, potřebujeme mít prostředky, které by nám takovéto nekonečné množiny umožnily specifikovat. Těmito prostředky jsou

- převodníky
- překladové gramatiky

## 3.1.1 Převodník

Převodník vznikne poměrně jednoduchým rozšířením definice automatu o možnost výstupu. V souvislosti s překladem se tedy můžeme setkat např. se zásobníkovým převodníkem. Ilustrujme si toto rozšíření na konečném automatu.

### 3.1.1.1 Konečný převodník

Jednoduchým rozšířením konečného automatu o možnost výstupu řetězu symbolů nad výstupní abecedou získáme tzv. *konečný převodník*.

**Definice 2:** Konečný převodník $M$ je šestice

$$M = (Q, \Sigma_I, \Sigma_O, g, q_0, F),$$

kde

$Q$ je konečná množina stavů,

$\Sigma_I$ je vstupní abeceda,

$\Sigma_O$ je výstupní abeceda,

$g$ je zobrazení $Q \times (\Sigma \cup \{\varepsilon\})$ do množiny konečných podmnožin $Q \times \Sigma_O^*$,

$q_0 \in Q$ je počáteční stav,

$F$ je konečná množina koncových stavů.

Obdobně jako u konečného automatu můžeme i zde definovat konfiguraci konečného převodníku.

**Definice 3:** *Konfigurace* konečného *převodníku* $M$ je trojice

$$(q, x, y) \in Q \times \Sigma_I^* \times \Sigma_O^*,$$

kde

$q$ je stav konečného převodníku

$x \in \Sigma_I^*$ je dosud nepřečtená část vstupního řetězce (sufix)

$y \in \Sigma_O^*$ je dosud vytvořená část výstupního řetězce (prefix)

Mějme dvě konfigurace konečného převodníku $M$, $(q, ax, y)$ a $(r, x, yz)$, $q, r \in Q$, $x \in \Sigma_I^*$, $y, z \in \Sigma_O^*$, $a \in \Sigma_I \cup \{\varepsilon\}$. $M$ přímo přejde z konfigurace $(q, ax, y)$ do konfigurace $(r, x, yz)$, $(q, ax, y) \Rightarrow (r, x, yz)$, když a jen když $g(q, a)$ obsahuje $(r, z)$.

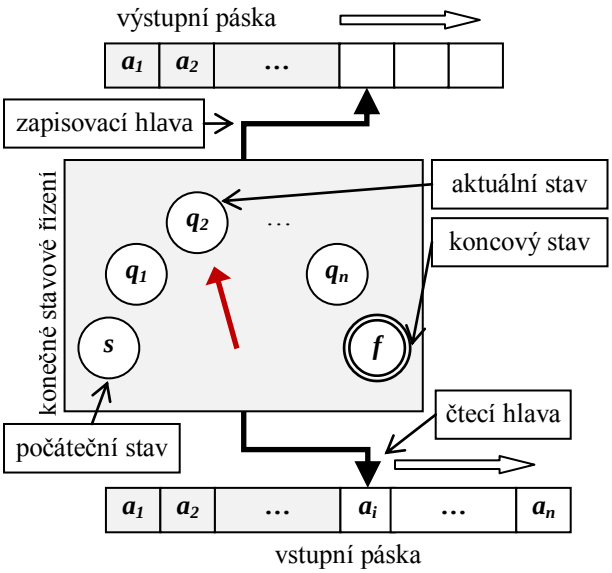
Pokud chceme zkrátit zápis, například přechod ze startovní do koncové konfigurace bez určení přesného počtu kroků, můžeme použít $(q, ax, y) \Rightarrow^* (r, x, yz)$. Přesný počet kroků můžeme zapsat obdobně, jen s tím rozdílem, že nahradíme znak $*$ číslem vyjadřujícím počet daných přechodů (např. $(q, ax, y) \Rightarrow^2 (r, x, yz)$).

Konfiguraci $(q_0, x, \varepsilon)$, $x \in \Sigma_I^*$ nazýváme *počáteční*, konfiguraci $(q, \varepsilon, y)$, $q \in F$, $y \in \Sigma_O^*$ nazýváme *koncovou*.

$$(q_0, x, \varepsilon) \Rightarrow^* (q, \varepsilon, y), q \in F$$

**Definice 4:** Překlad $P(M)$ definovaný konečným převodníkem $M$ je množina dvojic

$$P(M) = \{(x, y); (q_0, x, \varepsilon) \Rightarrow^* (q, \varepsilon, y), q \in F, x \in \Sigma_I^*, y \in \Sigma_O^*\}$$




**Obrázek 2: Konečný převodník**

### 3.1.2 Překladová gramatika

Překladové gramatiky jsou modifikací klasických gramatik, na rozdíl od nich však nemají na vstupu počáteční symbol, ale řetězec ze vstupního jazyka. Posloupností derivačních kroků dochází k jejímu postupnému překladu na větu z výstupního jazyka. Příklad překladové gramatiky bude uveden v kapitole 6.

# 4 Gramatiky s rozptýleným kontextem

Nyní se podívejme na gramatiky s rozptýleným kontextem (*Scattered Context Grammars*). V minulosti bylo provedeno mnoho pokusů o specifikace přirozených i programovacích jazyků pomocí syntaktické definice. Tato syntaktická definice by měla být taková, aby k ní mohly být připojeny i sémantické interpretace vět. Hlavním problémem a zároveň motivací k zavedení gramatik s rozptýleným kontextem ([3]) byla neexistence tříd gramatik dovolujících popsat uvedenou syntaktickou strukturu. Běžné bezkontextové gramatiky byly slabé, například nebylo možné definovat jazyk typu $L = \{ww | w \in \{a, b\}^*\}$, který vyžaduje předávání informací mezi vzdálenějšími částmi generované větné formy. Na druhou stranu kontextové gramatiky byly zbytečně silné a složité. Gramatiky s rozptýleným kontextem zavedené v roce 1969 S. Greibachovou a J. Hopcroftem ([3]) neodpovídají následující definici ([Definice 5:]), jelikož neobsahovaly vymazávací pravidla (tzv. $\varepsilon$-pravidla). V dnešní terminologii je nazýváme nevymazávajícími (nebo také nezkracujícími), viz. [Definice 7:]

**Definice 5:** *Gramatika s rozptýleným kontextem* (*Scattered Context Grammar*) je čtveřice

$$G = (V, T, P, S),$$

kde

$V$ je úplná abeceda

$N$ je konečná množina neterminálních symbolů $N \subset V$

$T$ je konečná množina terminálních symbolů, $T = V - N$

$S \in N$ je počáteční neterminál (startující symbol gramatiky $G$)

$P$ je konečná množina gramatických pravidel tvaru

$$(A_1, \dots, A_n) \to (x_1, \dots, x_n), n \geq 1, A_i \in N, x_i \in V^*, \forall\, 1 \leq i \leq n$$

**Definice 6:** Mějme větné formy

$$u = u_1 A_1 \dots u_n A_n u_{n+1},$$
$$v = u_1 x_1 \dots u_n x_n u_{n+1},$$

a pravidlo

$$p = (A_1, \dots, A_n) \to (x_1, \dots, x_n) \in P, u_i \in V^*\ \forall\, 1 \leq i \leq n + 1$$

Gramatika $G$ provede *derivační krok* z $u$ do $v$ podle pravidla $p$ což zapisujeme $u \Longrightarrow_G v\,[p]$, resp. $u \Longrightarrow_G v$ nemůže-li dojít k záměně použitého pravidla.

Obvyklým způsobem můžeme rozšířit derivační krok $\Longrightarrow$ na sérii $m$ derivačních kroků $\Longrightarrow^m$, pro celé číslo $m \geq 0$ určující počet těchto kroků. Také můžeme zavést $\Longrightarrow^+$, kdy je proveden alespoň

jeden derivační krok a $\Longrightarrow^*$, kdy nemusí být proveden žádný derivační krok (pokud není proveden žádný krok, musí být obě strany shodné).

V případě splnění podmínky $A_i \notin alph(u_i), \forall\, 1 \leq i \leq n$ nazýváme derivační krok nejlevějším ($u \;{}_{lm}\Longrightarrow_G v\;[p]$). Analogicky musí pro nejpravější derivační krok ($u \;{}_{rm}\Longrightarrow_G v\;[p]$) platit $A_i \notin alph(u_{i+1}), \forall\, 1 \leq i \leq n$.

Definujme:

$$lhs(p) = A_1, \dots, A_n$$

$$rhs(p) = x_1, \dots, x_n$$

$$len(p) = n$$

Pokud je délka pravidla $len(p) \geq 2$, nazýváme toto pravidlo *kontextové*. V opačném případě, tj. $len(p) = 1$, jde o pravidlo *bezkontextové*.

*Jazyk s rozptýleným kontextem* $L$ je jazyk generovaný gramatikou s rozptýleným kontextem $G$ pokud platí $L = L(G) = \{x: x \in T^*, S \Longrightarrow_G^* x\}$. Třídu jazyků s rozptýleným kontextem označujeme $\mathcal{L}(SC)$ a je shodná s třídou rekurzivně vyčíslitelných jazyků, $\mathcal{L}(SC) = \mathcal{L}(RE)$, viz. Theorem 3.20 v [7].

## 4.1 Nevymazávající gramatiky s rozptýleným kontextem

**Definice 7:** *Nevymazávající (nezkracující) gramatika s rozptýleným kontextem* (*Propagating Scattered Context Grammar*) je čtveřice

$$G = (V, T, P, S),$$

kde $V, T, S$ mají stejný význam jako v předchozí definici a $P$ je konečná množina pravidel tvaru

$$(A_1, \dots, A_n) \rightarrow (x_1, \dots, x_n), n \geq 1, A_i \in N, x_i \in V^+, \forall\, 1 \leq i \leq n$$

Třída jazyků generovaných nevymazávajícími gramatikami s rozptýleným kontextem se označuje $\mathcal{L}(PSC)$.

**Příklad 1:** Mějme gramatiky $G_1$ a $G_2$:

$G_1 = (\{S, A, a, b, c\}, \{a, b, c\}, P_1, S)$,

$P_1 = \{(S) \rightarrow (aAbAcA),\ (A, A, A) \rightarrow (aA, bA, cA),\ (A, A, A) \rightarrow (\varepsilon, \varepsilon, \varepsilon)\}$

$G_2 = (\{S, A, a, b, c\}, \{a, b, c\}, P_2, S)$,

$P_2 = \{(S) \rightarrow (AAA),\ (A, A, A) \rightarrow (aA, bA, cA),\ (A, A, A) \rightarrow (a, b, c)\}$

Řetězec $aaabbbccc$ tyto gramatiky vygenerují následovně:

$S \Longrightarrow_{G_1} aAbAcA \Longrightarrow_{G_1} aaAbbAccA \Longrightarrow_{G_1} aaaAbbbBcccC \Longrightarrow_{G_1} aaabbbccc$

$S \Longrightarrow_{G_2} AAA \Longrightarrow_{G_2} aAbAcA \Longrightarrow_{G_2} aaAbbAccA \Longrightarrow_{G_2} aaabbbccc$

Je zřejmé, že obě uvedené gramatiky definují stejný jazyk $L(G_1) = L(G_2) = \{a^n b^n c^n : n \geq 1\}$, který není bezkontextový. Rozdíl těchto gramatik spočívá v existenci vymazávajícího pravidla u gramatiky $G_1$.

**Definice 8:** Mějme gramatiku s rozptýleným kontextem $G = (V, T, P, S)$. *Stupeň kontextové citlivosti* gramatiky $G$ (*degrese of context sensitivity*) definujeme jako:

$$dcs(G) = |\{p : p \in P, lhs(p) \geq 2\}|.$$

*Maximální kontextová citlivost* gramatiky $G$ (*maximum context sensitivity*):

$$mcs(G) = \max(\{len(p) - 1 : p \in P\}).$$

Celková kontextová citlivost gramatiky $G$ (*overall context sensitivity*):

$$ocs(G) = len(p_1) + \cdots + len(p_n) - n, \text{ kde } P = \{p_1, \ldots, p_n\}.$$

### 4.1.1 Generativní síla

Jak již bylo zmíněno dříve, gramatiky s rozptýleným kontextem mají stejnou generativní sílu jako neomezené gramatiky, tj. dovolují popsat jazyky typu 0 Chomského klasifikace. Nyní se podívejme na generativní sílu nevymazávajících gramatik s rozptýleným kontextem.

Nejprve se zaměříme na vztah $\mathcal{L}(CF)$ a $\mathcal{L}(PSC)$. Jak je zřejmé z příkladu v předchozí kapitole (Příklad 1:) jsou vymazávající gramatiky s rozptýleným kontextem silnější než bezkontextové.

$$\mathcal{L}(CF) \subset \mathcal{L}(PSC)$$

Jelikož $PSC$ gramatiky neobsahují vymazávající pravidla, mohou být jejich derivace simulovány kontextovými gramatikami. Proto tedy platí

$$\mathcal{L}(PSC) \subseteq \mathcal{L}(CS)$$

## 4.2 *k*-omezené vymazávání

Vzhledem k rozdílné generativní síle $PSC$ a $SC$ gramatik, není možné převést libovolnou gramatiku s rozptýleným kontextem na ekvivalentní gramatiku bez vymazávajících pravidel.

Nyní se podíváme na speciální podmínku, při jejímž splnění tuto konverzi bude možno provést. Touto podmínkou je tzv. k-omezené vymazávání. Hlavní myšlenkou *k*-omezeného vymazávání je podmínka, že mezi dvěma neterminálními symboly, z nichž gramatika $G$ vyderivuje neprázdný řetězce, se vyskytuje nejvýše *k* neterminálních symbolů, z nichž je vyderivován prázdný řetězec.

$$\begin{array}{ccccccc} X_1 & Y_1 & \dots & Y_n & X_2 \\ \Downarrow & \Downarrow & \Downarrow & \Downarrow & \Downarrow \\ w_1 & \underbrace{\varepsilon \ \dots \varepsilon}_{\max k} & & & w_2 \end{array}$$

$X_i, Y_i \in N, w_1, w_2 \in T^+$.

**Definice 9:** *Bázová (základní) gramatika* (*core grammar*) gramatiky s rozptýleným kontextem $G = (V, T, P, S)$, označovaná jako $core(G)$ je bezkontextová gramatika $core(G) = (V, T, P', S)$, kde množina pravidel $P' = \{A_i \to x_i : (A_1, \dots, A_i, \dots, A_n) \to (x_1, \dots, x_i, \dots, x_n) \in P, 1 \leq i \leq n\}$.

**Definice 10:** Nechť $G = (V, T, P, S)$ je $SC$ gramatika a nechť $core(G)$ je její bázová gramatika. Mějme derivaci $S \Rightarrow_G^* x$ ve formě $S \Rightarrow_G^* uAv \Rightarrow_G^* x$. Mějme $cf(S \Rightarrow_G^* x)$ jako bezkontextovou simulaci pravidla $S \Rightarrow_G^* x$ (definice viz. Definition 4.11 v [7]). Dále mějme derivační strom $t$ odpovídající derivaci $S \Rightarrow_{core(G)}^* x$ a uvažujme podstrom tohoto derivačního stromu, jehož kořenem je $A$. Pokud hranice tohoto podstromu je $\varepsilon$, pak $G$ vymaže $A$ v $S \Rightarrow_G^* uAv \Rightarrow_G^* x$. Tuto skutečnost symbolicky zapíšeme jako

$$\grave{A}.$$

V opačném případě $G$ v $S \Rightarrow_G^* uAv \Rightarrow_G^* x$ symbol $A$ nevymaže, což značíme

$$\acute{A}.$$

Pokud v $w = \grave{A}_1, \dots, \grave{A}_n$ nebo $w = \acute{A}_1, \dots, \acute{A}_n$ pro $n \geq 1$, píšeme $\grave{w}$, resp. $\acute{w}$.

**Definice 11:** Mějme $SC$ gramatiku $G = (V, T, P, S)$ a číslo $k \geq 0$. $G$ *vymaže své neterminální symboly obecným k-omezeným způsobem*, pokud pro každý řetězec $y \in L(G)$ existuje derivace $S \Rightarrow_G^* y$ v níž každá větná forma $x$ splňuje následující podmínky:

1, pro všechny $x = uAvBw$, kde $\acute{A}, \acute{B}, \grave{v}$ platí $|v| \leq k$

2, pro všechny $x = uAvBw$, kde $\acute{A}$ platí: pokud $\grave{u}$ pak $|u| \leq k$ a pokud $\grave{w}$ pak $|w| \leq k$

Třídu jazyků generovaných $SC$ gramatikami, které vymazávají neterminální symboly $k$-omezeným způsobem, značíme $\mathcal{L}(SC, \varepsilon, k)$.

Pro ilustraci může posloužit příklad Example 4.14 z [7]. Uvedený příklad ilustruje, jak užitečné může být vymazávání při ověřování vztahů mezi jednotlivými podřetězci větné formy.

**Teorém 1:** Ke každé $SC$ gramatice $G$, která vymazává neterminální symboly $k$-omezeným způsobem, existuje $PSC$ gramatika $\bar{G}$, pro niž platí $L(\bar{G}) = L(G)$.

Princip důkazu: Gramatika $\bar{G}$ simuluje gramatiku $G$ pomocí neterminálních symbolů speciálního tvaru $\langle \dots \rangle$. V každém z těchto symbolů je v každém kroku derivace zaznamenán řetězec odpovídající větné formy gramatiky $G$. Podrobnosti k důkazu viz. strana 60 v [7].

# 5 Generátory větných rozborů

V této kapitole se budu věnovat jedné z nejdůležitějších částí analýzy a zpracování jazyků, kterou jsou rozbory (*parses*). Z hlediska teorie formálních jazyků je rozbor posloupnost gramatických pravidel aplikovaných od počátku derivace větné formy až po její úspěšné dokončení. Obvykle je získání této informace jedním z cílů syntaktické analýzy. V této kapitole budou prezentovány některé zajímavé poznatky o využití gramatik s rozptýleným kontextem k větnému rozboru.

**Definice 12:** Pro každou $SC$ gramatiku $G = (V, T, P, S)$ existuje množina návěští pravidel, označovaná jako $lab(G)$, $|lab(G)| = |P|$. Navíc existuje taková bijekce z $P$ do $lab(G)$, která přiřazuje každému pravidlu $(A_1, \dots, A_n) \rightarrow (x_1, \dots, x_n) \in P$ návěští $l \in lab(G)$. Říkáme, že pravidlo $(A_1, \dots, A_n) \rightarrow (x_1, \dots, x_n)$ je označeno návěštím $l$.

**Definice 13:** Pokud každý derivační krok úspěšné derivace v $SC$ gramatice $G$ je Nejlevější (nejpravější), pak $G$ generuje $L(G)$ nejlevějším (nejpravějším) způsobem.

**Definice 14:** Mějme $SC$ gramatiku $G = (V, T, P, S)$ a úspěšnou derivaci $S \Rightarrow_G^* x\ [\rho]$, kde $x \in T^*$, $\rho \in lab(G)^*$. $x$ je tedy věta generovaná $G$ pomocí rozboru $\rho$. Nechť $lab(G) \subseteq T$.

$G$ je *vlastní generátor vět s jejich rozbory*, pokud platí:

$$L(G) = \{x : x = y\rho, y \in (T - lab(G))^*, \rho \in lab(G), S \Rightarrow_G^* x\ [\rho]\}.$$

Pokud navíc $G$ generuje $L(G)$ nejlevějším (nejpravějším) způsobem, pak $G$ je *vlastní nejlevější (nejpravější) generátor vět s jejich rozbory.*

Obdobně můžeme definovat *vlastní generátor vět, které předcházejí jejich rozbory* (analogicky platí i pro nejlevější a nejpravější):

$$L(G) = \{x : x = \rho y, y \in (T - lab(G))^*, \rho \in lab(G), S \Rightarrow_G^* x\ [\rho]\}.$$

**Příklad 2:** Mějme $SC$ gramatiku $G = (V, T, P, S)$:

$V = \{S, A, B, C, \$, a, b, c, 1,2,3,4\}$

$T = \{a, b, c, 1,2,3,4\}$

$P = \{\ 1: (S) \rightarrow (1),$

$2: (S) \rightarrow (ABC2\$),$

$3: (A, B, C, \$) \rightarrow (aA, bB, cC, 3\$),$

$4: (A, B, C, \$) \rightarrow (a, b, c, 4)\ \}$

Je zřejmé, že $L(G) = \{a^n b^n c^n \rho : n \geq 0, S \Rightarrow_G^* a^n b^n c^n \rho\ [\rho]\}$ a každý derivační krok je nejlevější i nejpravější, tedy $G$ je *vlastní nejlevější i vlastní nejpravější generátor vět s jejich rozbory*.

# 5.1 Obecné generátory

Nyní se podívejme na nejzajímavější vlastnost vlastních generátorů vět s jejich rozbory. Tuto vlastnost popisuje následující teorém.

**Teorém 2:** Pro každý rekurzivně vyčíslitelný jazyk $L \in L(RE)$ existuje $PSC$ gramatika $G$ taková, že $G$ je vlastním generátorem vět s jejich rozbory a $L = L(G)\ //\ lab(G)^+$.

Důkaz lze nalézt v kapitole 7.2 z [7].

**Teorém 3:** Pro každý rekurzivně vyčíslitelný jazyk $L$ existuje $PSC$ gramatika $G$ taková, že $G$ je vlastním generátorem vět které přecházejí jejich rozbory a $L = lab(G)^+ \backslash\backslash L(G)$.

Toto tvrzení je analogické předchozímu

**Teorém 4:** Pro každý rekurzivně vyčíslitelný jazyk $L$ existuje $PSC$ gramatika $G$ taková, že $G$ je vlastním generátorem vět které přecházejí jejich rozbory a $L = \{\$\}^+ \backslash\backslash L(G)$.

Jde o přímý důsledek předchozího teorému, kdy místo návěští provedených derivačních kroků generujeme symbol $.

# 5.2 Kanonické generátory

Podívejme se na speciální variantu generátorů. Pokud budeme uvažovat vlastní nejlevější (nejpravější) generátor vět s jejich rozborem, dostaneme podobní výsledky jako u obecných generátorů s jedním významným rozdílem. Derivace provedená nejlevějším (nejpravějším) generátorem je zopakovatelná, protože víme, že byly vždy nahrazeny nejlevější (nejpravější) neterminální symboly.

**Teorém 5:** Pro každý rekurzivně vyčíslitelný jazyk $L$ existuje $PSC$ gramatika $G = (\bar{V}, \bar{T}, P, S)$ taková, že $G$ je vlastním nejlevějším (nejpravějším) generátorem vět s jejich rozbory, $|\bar{V} - \bar{T}| \leq 6$ a $L = L(G)//lab(G)^*$ ($L$ získáme z $L(G)$ eliminací všech návěští pravidel).

Podrobný důkaz lze nalézt v kapitole 7.3 knihy [7].

# 6 Překladové gramatiky s rozptýleným kontextem

Jak již bylo uvedeno dříve, překladové gramatiky jsou modifikací klasických gramatik, na rozdíl od nich však nemají na vstupu počáteční symbol, ale řetězec ze vstupního jazyka. Posloupností derivačních kroků dochází k jejímu postupnému překladu na větu z výstupního jazyka. Tuto modifikaci můžeme aplikovat i na gramatiky s rozptýleným kontextem.

**Definice 15:** *Překladová gramatika s rozptýleným kontextem* je čtveřice

$$G = (V, T, P, I),$$

kde

$V$ je úplný slovník

$T \subset V$ je konečná množina terminálních symbolů (výstupní slovník)

$P$ je konečná množina $SC$ gramatických pravidel

$I \subset$ V je vstupní slovník

Definice derivačního kroku překladové $SC$ gramatiky je shodná s definicí běžné $SC$ gramatiky (Definice 6:)

**Definice 16:** *Překlad* $T$ definovaný překladovou gramatikou $G$ z $K \subseteq I^*$ označujeme $T(G, K)$ a definujeme jej jako:

$$T(G, K) = \{(x, y): x \Rightarrow_G^* y\ , x \in K, y \in T^*\}$$

$x$ nazýváme vstupní větou, $y$ pak výstupní větou.

**Příklad 3:** Mějme překladovou $SC$ gramatiku $G = (V, T, P, I)$

$V = \{A, B, C, a, b, c\}$

$T = \{a, b, c\}$

$I = \{A, B, C\}$

$P = \{(A, B, C) -> (aa, b, cc)\}$

Za vstupní větu zvolme $AAABBBCCC$.

$AAABBBCCC \Rightarrow_G aaAAbBBccCC \Rightarrow_G aaaaAbbBccccC \Rightarrow_G aaaaaabbbcccccc$,

tedy $AAABBBCCC \Rightarrow_G^* aaaaaabbbcccccc$

$$(AAABBBCCC, aaaaaabbbcccccc) \in T(G, I^*)$$

Pokud omezíme vstupní věty do jazyka $L = \{A^n B^n C^n : n \geq 1\}$, dostaneme

$$T(G, L) = \{(A^n B^n C^n, a^{2n} b^n c^{2n}): n \geq 1\}$$

A každá věta $A^n B^n C^n$ je přeložena na odpovídající větu $a^{2n} b^n c^{2n}$.

# 7 Závěr

Tato práce byla věnována modifikacím gramatik s rozptýleným kontextem využitelným při překladu programovacích i přirozených jazyků

Byly zde prezentovány gramatiky s rozptýleným kontextem v základní variantě i variantě nevymazávající, která je za běžných podmínek slabší. Pozornost si zde zasluhuje také zavedení speciální podmínky pro gramatiky s rozptýleným kontextem, k-omezené vymazávaní, při jejímž splnění je možné obecnou gramatiku s rozptýleným kontextem převést na nevymazávající variantu.

Následně byly ukázány některé modifikace uvedených gramatik, využitelné v překladu jazyků (a to jak programovacích, tak i přirozených). Nejprve šlo o tzv. generátory. Ty mají velice zajímavou vlastnost – ke každému rekurzivně vyčíslitelnému jazyku existuje nevymazávající gramatika s rozptýleným kontextem, která dovoluje generovat věty z tohoto jazyka následované jejich rozborem. Po odstranění rozborů z vět generovaných pomocí generátorů získáváme rekurzivně vyčíslitelný jazyk pomocí mnohem slabšího prostředku.

Modifikace uplatnitelná především v překladu přirozených jazyků je překladová gramatika s rozptýleným kontextem. Jako jeden z mnoha příkladů použití může být v anglickém jazyce velmi jednoduchá transformace oznamovací věty na otázku a naopak (využití v anglickém jazyce je na mnoha příkladech ilustrováno v [7]).

# Literatura


[1] **Aho, A. V., Ullman, J. D.:** *The Theory of Parsing, Translation and Compiling - Volume I: Parsing*, Englewood Cliffs, New Jersey, Prentice-Hall, 1972, ISBN: 0-13-914556-7.

[2] **Aho, A. V., Ullman, J. D.:** *The Theory of Parsing, Translation and Compiling - Volume II: Compiling*, Englewood Cliffs, New Jersey, Prentice-Hall, 1973, ISBN: 0-13-914564-8.

[3] **Greibach, S., Hopcroft, J.:** Scattered Context Grammars, *Journal of Computer and System Sciences*, 1969, pp. 233-247.

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

[5] **Meduna, A.:** *Elements of Compiler Design*, New York, Taylor & Francis, 2008, ISBN: 1-4200-6323-5.

[6] **Meduna, A., Lukáš, R.:** *Formální jazyky a překladače,* studijní opora, Brno, FIT VUT v Brně, 2006.

[7] **Meduna, A., Techet,J.:** *Scattered Context Grammars and Their Applications*, WIT Press, 2010, ISBN: 978-1-84564-426-0.

[8] **Wikipedia contributors:** Chomsky hierarchy, [Online] Wikipedia, The Free Encyclopedia, 21. říjen 2009 [Citace: 4. únor 2010] http://en.wikipedia.org/w/index.php?title=Chomsky_hierarchy&oldid= 321254719