# Syntaktická analýza jazyků generovaných řízenými gramatikami

Martin Hrubý

7. ledna 2002

**Abstrakt**

Tento dokument vznikl jako doprovodný text semináře kurzu Moderní teoretická informatika. Pojednává o principech řízených gramatik a o možnostech jejich zpracování. Zabývá se především jedním konkrétním typem řízené gramatiky obecně známé pod názvem Cooperating Distributed Grammar. Byla navržena jednoduchá metoda syntaktické analýzy CD gramatik, dokument popisuje teoretické předpoklady pro analyzovatelnost CDG a prototyp syntaktického analyzátoru CDG.

**Klíčová slova:** Cooperating Distributed Grammar, syntaktická analýza, levá derivace, překladače

## 1 Úvod

V kurzu Moderní teoretická informatika jsme se seznámili se skupinou gramatik, které se nazývají *řízené gramatiky*. Zjednodušeně můžeme popsat princip těchto gramatik tak, že jsou to gramatiky s bezkontextovými pravidly, které určitým způsobem (řízením přepisování) dosáhnou generovací síly větší, než je síla obyčejných bezkontextových gramatik.

*Bezkontextové gramatiky* tvoří významnou skupinou gramatik podle Chomského. Jsou to nejčastěji prakticky používané gramatiky, protože syntaktická analýza bezkontextových jazyků je jednoduchá a průhledná. Stačí na ni stroj s jedním zásobníkem. Tyto analyzátory se velmi snadno implementují v programovacích jazycích a v podstatě bezkontextové gramatiky postačí v drtivé většině prakticky řešených aplikací gramatik a překladačů. Myslím tím, že při návrhu gramatiky pro určitý programovací jazyk, se obvykle vystačí s rozsahem bezkontextových gramatik a pravidel. Občas se ale (a to snad výjimečně) objeví situace, kdy určitou jazykovou konstrukci opravdu nelze zapsat v bezkontextových přepisovacích pravidlech. Pak se prohlásí, že bezkontextové jazyky jsou slabé a nedostačující. Musíme se však ptát, na co jsou bezkontextové jazyky slabé. Jsou slabé při pokusech v praktickém využití nebo jsou slabé *jen ze své teoretické podstaty* a v porovnání s vyššími třídami jazyků?

V posledních letech probíhá výzkum gramatických systémů a teorií gramatik s cílem přinést gramatiku s vyšší generovací silou, která zároveň udržuje jednoduchost a lidskou přístupnost bezkontextových pravidel. Proto vznikly gramatické systémy a řízené gramatiky. Je jich celkem rozsáhlá skupina a všechny v podstatě řeší stejný problém: jak ve přepisovacích pravidlech a v relaci derivace skrýt a zamaskovat *kontext.*

# 2 Cooperating Distributed Grammar (CDG)

Bývá nesnadné přeložit do našeho jazyka anglické technické termíny, tady lze název *Cooperating Distributed Grammars* (CDG) přeložit na *splupracující rozložené gramatiky* a snad to vystihuje i jejich princip. Začneme jejich definicí podle ([1,2]):

CDG je konstrukce:

$$G = (N, T, S, P_1, P_2, \dots, P_k) \tag{1}$$

kde $N, T, S$ mají obvyklý význam a $P_i, 1 \leq i \leq k$ jsou konečné množiny bezkontextových pravidel nad $N \cup T$, což znamená tvaru $A \rightarrow (N \cup T)^*$, kde $A \in N$. Množiny $P_i$ jsou nazývány *komponenty gramatiky* $G$. Písmenem $k$ budu označovat počet komponent v gramatice.

Princip generování vět z CDG popíši nejdříve neformálně. Kontext generování je vždy rozpracovaná větná forma a číslo jedné aktivní komponenty, ostatní komponenty se nezahrnují do přepisování. V daném kontextu lze aplikovat na přepisování pouze pravidla aktivní komponenty a komponenta je aktivní, dokud existuje aplikovatelné pravidlo aktivní komponenty. Není-li už v aktivní komponentě aplikovatelné pravidlo, aktivitu přebere jiná libovolná komponenta. Uvidíme, že tento princip je klíčový pro CDG.

Formálně ([1]) je jazyk generovaný CDG gramatikou v *t-módu*:

$$L_t(CDG) = \{w | w \in T^*, S \Rightarrow^t_{P_{i_1}} w_1 \Rightarrow^t_{P_{i_2}} \dots \Rightarrow^t_{P_{i_m}} w_m = w, m \geq 1, 1 \leq i_j \leq k, 1 \leq j \leq m\} \tag{2}$$

**Příklad 1:**

$$G = (\{S, A, C, X, Y\}, \{a, b, c\}, S, P_1, P_2, P_3) \tag{3}$$

kde

$P_1 = \{S \rightarrow AC, X \rightarrow A, Y \rightarrow C\}$,
$P_2 = \{A \rightarrow aXb, C \rightarrow cY\}$,
$P_3 = \{A \rightarrow ab, C \rightarrow c\}$

ukázka generování věty

$$S \Rightarrow AC \Rightarrow aXbC \Rightarrow aXbcY \Rightarrow aAbcY \Rightarrow aAbcC \Rightarrow aabbcC \Rightarrow aabbcc$$

Tato gramatika (3) generuje jazyk $\{a^n b^n c^n \mid n \geq 1\}$.

**Poznámka:** všimněme si nedeterminismu gramatiky způsobeného pravidly $A_2$ a $A_3$. C-pravidla nedeterminismus ale v podstatě nezpůsobují, protože o výběru C-pravidla rozhodne kontext zavedený A-pravidlem.

## 2.1 Modifikovaná CDG (mCDG)

**Demonstrační gramatika (již deterministická)**

$$G = (\{S, A, C, X, Y), \{a, b, c\}, S, P_1, P_2, P_3) \tag{4}$$

$P_1 = \{S \to AC, X \to A, Y \to C\}$
$P_2 = \{A \to aXb, C \to cY\}$
$P_3 = \{A \to d, c \to d\}$

Tato gramatika (4) generuje jazyk $\{a^n db^n c^n d | n \geq 1\}$.

**Úvodní definice**

Zatím si nepovolíme výskyt e-pravidel, protože to celkem komplikuje následující definice. Pro navigaci gramatikou zavedeme návěští pravidel (labels). Komponentu budeme brát jako seřazenou posloupnost pravidel. Pak můžeme psát: $P_{i,j} : A$ je j-té pravidlo i-té komponenty.

Kontext derivace (větná forma) je $V^+$, kde

$$V \in (T \cup (N \times Z^*)) \tag{5}$$

$$Z = \{1..k\} \tag{6}$$

$Z^*$ je posloupnost (řetězec) čísel, které budou znamenat čísla komponent.

**Příklady větných forem:**

$abAbC$, $abAbC(1,2)$

Pro další definice je nutné upřesnit relaci derivace mCDG gramatiky a zavést *levou derivaci*. Je to komplikované, protože se do větné formy za nejlevějším nonterminálem musí zaznačit odložené přepsání a zachovat tak kontext mezi pravidly.

**Definice 1: Aplikace čísla komponenty na větnou formu**

Funkce $apply : V^* \times Z \to V^*$ aplikuje číslo komponenty $i$ na větnou formu $w$ tak, že:

$apply(e, i) = e$
$apply(ab, i) \bigwedge (a \in T) = a.apply(b, i)$
$apply(ab, i) \bigwedge (a = N_x() \bigwedge N_x \in P_i) = a(i).apply(b, i)$
$apply(ab, i) \bigwedge (a = N_x(list)) = a(list, i).apply(b, i)$

Pro zápis volání funkce *apply(w,n)* zavedu notaci $w[n]$. Aplikaci řetezce čísel na větnou formu $w[a_1, ..., a_b]$ je $(((w[a_1])[a_2])...)[a_b]$, tedy se postupně aplikují čísla $a_1$ až $a_b$.

**Příklad 2:** (používá gramatiku (4))

$abc[1] = abc$
$abC[1] = abC$
$abC(2,1)[2] = abC(2,1,2)$

**Definice 2: Levá derivace v mCDG**

Levá derivace $v_1 \Rightarrow_L v_2$; $v_1, v_2 \in V^+$, $\alpha \in T^*$, $\beta \in V^*$ je

1. $AC \Rightarrow_L aXbC(2)$ (příklad)
   $v_1 = \alpha A \beta$ a existuje $i, j : P_{i,j} : A \to \gamma$, pak $v_2 = \alpha\gamma[i]\beta[i]$
2. $aadbbcC(2,1,3) \Rightarrow_L aadbbccY(2,1,3)$
   $v_1 = \alpha A(i, d_1, .., d_b)\beta$ a existuje $j : P_{i,j} : A \to \gamma$, pak $v_2 = \alpha\gamma[i, d_1, ..., d_b]\beta[i]$
3. $aadbbcY(2,1,3) \Rightarrow_L aadbbccY(1,3)$
   $v_1 = \alpha A(i, d_1, ..., d_b)\beta$ a neexistuje $j : P_{i,j} : A \to \gamma$, pak $v_2 = \alpha A(d_1, ..., d_b)\beta[i]$

Případ 1 modeluje přepnutí kontextu z jedné komponenty na druhou, druhý případ definuje vynucené použití pravidla z konkrétní komponenty a poslední případ je ignorování kontextu a přechod na další.

Existence levé derivace je klíčová pro deterministickou syntaktickou analýzu (odkazy). Následuje ukázka generování věty levou derivací (zjednodušeně):

---

S $\Rightarrow_L AC \Rightarrow_L aXbC(2) \Rightarrow_L aAbC(2,1) \Rightarrow_L aaXbbC(2,1,2) \Rightarrow_L aaAbbC(2,1,2,1) \Rightarrow_L$ $aadbbC(2,1,2,1,3) \Rightarrow_L aadbbcY(1,2,1,3) \Rightarrow_L aadbbcC(2,1,3) \Rightarrow_L aadbbccY(1,3) \Rightarrow_L$ $aadbbccC(3) \Rightarrow_L aadbbccd$

---

**Věta:** Mějme CDG gramatiku $CDG = (N, T, S, P_1, ..., P_k)$, jazyk $L_l(CDG) = \{w \in T^* | S \Rightarrow_L^* w\}$ a jazyk $L_t$ podle (2). Pak platí:

$$L_l(CDG) = L_t(CDG) \tag{7}$$

**Komentář:** Jinými slovy platí, že derivace v t-módu a levá derivace jsou ekvivalentní. Tato věta vyžaduje provedení formálního důkazu, ten by však byl nad rámec této práce.

**Poznámka:** Určitými aspekty levé derivace v řízených gramatikách se zabývá [6].

**Definice 3:** Modifikovaná CD gramatika (mCDG) je konstrukce (1) s relací levé derivace.

## 2.2 Determinismus v mCDG

Determinismus je vlastnost stroje, kdy stroj v každém stavu své činnosti dokáže jednoznačně, podle aktuálního stavu a vstupních podmínek, rozhodnout o svém následujícím stavu. To znamená, že jsou přípustné pouze dvě situace: (1) není definován následující stav a pak nastane kritická chyba nebo (2) stav je definován a pak stroj (výpočet) přejde do následujícího stavu.

Stejně je to se syntaktickou analýzou. Syntaktická analýza věty je rekonstrukce posloupnosti derivací, které vedly k získání této věty. A mohlo by se stát, že: $S \Rightarrow^* \alpha A w$ a existuje více pravidel $A \rightarrow a_1|a_2|...|a_b$. Pak musí existovat pravidla, podle nichž analyzátor rozhodne větvení syntaktické analýzy.

Determinismus je v případě CDG dvojí:

**statický** nebo taky "determinismus na první pohled". Je stejný, jako posuzování bezkontextové gramatiky, když u CDG sloučíme všechna pravidla dohromady.

**dynamický** v případě CDG je běžné, že se určitý nonterminál musí přepsat v kontextu komponenty a kontext vznikne až při derivování (jako nonterminál $C$ v příkladu 1). Tady je analýza determinismu mnohem složitější (odkazy na mitranu).

### Situace u bezkontextových LL(1) gramatik

LL(1) je deterministická, pokud pro všechny nonterminály $A \in N$, platí (nejsou e-pravidla):

$$A \rightarrow a_1|a_2|...|a_b \in P \Rightarrow \forall i,j \in \{1..b\} : i \neq j : FIRST(a_i) \bigcap FIRST(a_j) = \emptyset \qquad (8)$$

Výpočet množiny FIRST je obecně známá věc (viz skripta, odkaz). V případě CDG je výběr pravidla omezován kontextem derivace (aktuální komponentou v t-módu). Proto zavádím pojem *globální FIRST* parametrizovaný navíc kontextem.

# 3 Syntaktická analýza mCDG

Syntaktická analýza je činnost inverzní ke generování vět. Řeší ji určitý stroj, což je v podstatě algoritmus, který v případě věty z daného jazyka skončí v konečném stavu a v případě jiné věty havaruje. Druhý úkol (pro překladače) je rekonstruovat posloupnost derivací vedoucích k zadané větě.

CDG jsou v podstatě bezkontextové gramatiky se skrytým kontextem. Proto jsem se snažil na ně použít metody analýzy bezkontextových jazyků rozšířené o řízení kontextem. Chtěl jsem zachovat filosofii těchto gramatik a tou je především vypadat na první pohled bezkontextově. Omezil jsem se na analýzu rekurzivním sestupem, jak se programuje v případě LL(1) gramatik. Je zřejmé, že pro analýzu CDG potřebujeme stroj na úrovni Turingova stroje nebo jeho ekvivalent-dvojzásobníkový automat. Síla Turingova stroje je v jeho pásce, tedy v posloupnosti informací s neomezeným přístupem. U dvojzásobníkového automatu je přesunutí z vrcholu jednoho zásobníku na druhý ekvivalentní posunu čtecí hlavy Turingova stroje.

## 3.1 Syntaktická analýza a překlad bezkontextových gramatik rekurzivním sestupem

K obyčejné bezkontextové LL(1) gramatice je velmi snadné napsat analyzátor. Ke každému pravidlu tvaru: $A \rightarrow a_1|a_2|...|a_N$ (tedy ke každému nonterminálu) se vygeneruje procedura:

```
int alex; // aktualne nacteny vstup z lexikatoru
void A() // procedura nonterminalu A
{
  if (alex in FIRST(a1)) { a1vf; }
  else
  if (alex in FIRST(a2)) { a2vf; }
  else
  ...
  if (alex in FIRST(aN)) { aNvf; }
  else
    error();
}
```

Sekvence *a1vf* až *aNvf* se nahradí posloupností příkazů přijetí terminálu *expect(a)* a volání procedury nonterminálu. Je-li $a_ivf = b_1b_2...b_j$, pak se posloupnost příkazů sestaví v pořadí 1 až $j$ tak, že pro každý $b_r, r = 1, 2, ..., j$:

**code**($b_r$)=

expect($b_r$) je-li $b_r \in T$

call($b_r$) je-li $b_r \in N$, volání procedury odpovídající nonterminálu $b_r$

Vykonávání takového programu sleduje levou derivaci a levý rozklad. Stav analýzy je dán aktuální procedurou (stav) a stavem zásobníku pro návraty z rekurzivního sestupu. To definuje i sílu analyzátoru a gramatiky - síla je ve třídě zásobníkového automatu.

## 3.2 Syntaktická analýza CDG rekurzivním sestupem

Snažil jsem se zachovat stejný styl analyzátoru, jako v předešlé kapitole. Rozdíl je ale podstatný: je nutné dopředu rozpracovávat větnou formu napravo od aktuálně přepisovaného nonterminálu (v programu: od aktuální procedury), což je definováno v levé derivaci.

Pro snadnou implementovatelnost analyzátoru v programovacích jazycích je základ stejný: rekurzivní sestup sledující levou derivaci, zbytek větné formy sledující průchody různými komponentami je umístěn explicitně v určité datové struktuře s charakterem seznamu (pásky).

# 4 Konstrukce překladače

Analyzátor je složen z těchto částí

**datová** obsahuje kompletní předpis gramatiky (je potřeba pro jádro)

**jádro** obsahuje funkce pro řízení analýzy

**překladač** z gramatiky vygenerovaný soubor procedur

## 4.1 Jádro analyzátoru

Jádro analyzátoru tvoří tyto funkce pracující nad globální datovou strukturou *větná forma (vf)*

**apply(w,n)** aplikuje číslo komponenty na větnou formu (podle definice 1)

**insert(w,i)** simuluje derivaci větné formy $A(list)\beta \Rightarrow_L^{P_{i,j}} \gamma[list]\beta[list]$

**call()** zprostředkovává volání procedury překladače.

$vf = A()\beta \Rightarrow$ `volej proceduru` $A$ `s parametrem` $0$, `A(0)`
$vf = A(i, tail)\beta \wedge A$ `existuje v` $P_i \Rightarrow$ `volej` $A(i)$
$vf = A(i, tail)\beta \wedge A$ `neexistuje v` $P_i \Rightarrow vf := A(tail)\beta$ `a znovu call`

## 4.2 Generování analyzátoru

Každý nonterminál gramatiky je v analyzátoru zastoupen jednou procedurou. Procedure zahrnuje všechna přepisovací pravidla nonterminálu ze všech komponent. To znamená, že pro každý nonterminál $A$ se vygeneruje procedura:

```
void A(int n, Attr attrs) {
  if (n==0) allA;
  else
  if (n==1) allA1;
  else
  ...
  if (n==k) allAk;
}
```

s parametry:

**int n** bude obsahovat číslo komponenty (číslovány od jedné) nebo nulu (výběr přes všechny komponenty)

**Attr attrs** bude obsahovat zděděné atributy

Každé přepisovací pravidlo i-té komponenty $A \to w,\ w = a_1 a_2 ... a_b$ se převede na

```
insert(w,i);
code(a1);
code(a2);
...
code(ab);
```

kde $code(a_r)$ je

expect($a_r$) je-li $a_r \in T$

call() je-li $a_r \in N$

**Poznámka:** V příloze (A) je kompletní program analyzátoru gramatiky (4).

## 4.3 Funkce analyzátoru[1]

Naznačím činnost analyzátoru na příkladu gramatiky (4). Počáteční stav je řetězec $S$, volá se procedura `S(0)`. Vývoj programu je naznačen v tabulce:

| vstupní řetezec | větná forma | volá se | poznámka | řízení |
|---|---|---|---|---|
| aadbbccd | S | S(0) | | |
| | AC | A(0) | gFIRST(A,0) | aXb |
| | aX(2)bC(2) | expect(a) | | |
| adbbccd | X(2)bC(2) | | X(2) není | |
| | XbC(2) | X(0) | | |
| | A(1)bC(2,1) | | A(1) ne | |
| | AbC(2,1) | A(0) | gFIRST(A,0) | aXb |
| | aX(2)bbC(2,1,2) | expect(a) | | |
| dbbccd | X(2)bbC(2,1,2) | | X(2) ne | |
| | XbbC(2,1,2) | X(0) | | |
| | AbbC(2,1,2,1) | | gFIRST(A,0) | d |
| | dbbC(2,1,2,1,3) | expect(d) | | |
| bbccd | bbC(2,1,2,1,3) | | vrací se k #1 | |
| | | expect(b) | vrací se k #1 | |
| bccd | bC(2,1,2,1,3) | expect(b) | | |
| ccd | C(2,1,2,1,3) | C(2) | | |
| | cY(1,2,1,3) | expect(c) | | |
| cd | Y(1,2,1,3) | Y(1) | | |

[1]viz příloha A

| | | | | |
|---|---|---|---|---|
| cd | C(2,1,3) | C(2) | | |
| | cY(1,3) | expect(c) | | |
| d | Y(1,3) | Y(1) | | |
| d | C(3) | C(3) | | |
| d | d | expect(d) | | |
| $\epsilon$ | $\epsilon$ | | accepted | |

V bodě "navrat bod #1" se vrací řízení programu po návratu z volání *call* (fakticky z call(X)) a program pokračuje příkazem expect(b). Je to místo, kde se spojuje řízení jednozásobníkového automatu (rekurzivní sestup) s paměťovým médiem *páska*. Podobně je to v místě mezi voláním nonterminálu A a C ($S_1 \rightarrow AC$).

# 5 Závěr

Byla navržena metoda syntaktické analýzy CD gramatik. Funkčnost metody byla ověřena prakticky na příkladu gramatiky (4). Pro úplnost této metody by bylo ještě třeba formálně dokázat, že definovaná levá derivace generuje stejné věty, jako obyčejná derivace v t-módu. Dále by bylo vhodné navrhnout algoritmus výpočtu množiny FIRST (gFIRST) a algoritmus ověření determinismu gramatiky (v dynamickém módu).

Výhoda navržené metody je zcela zřejmě v její jednoduché implementovatelnosti. Možná nevýhoda je snad v paměťové náročnosti při budování seznamu odložených derivací, tady je další bod pro optimalizace a zdokonalování.

# Reference


[1] Dassow, J., Paun, Gh., Rozenberg, G.: Grammar Systems, kapitoly 3-4

[2] Češka, M., Hruška, T.: Gramatiky a jazyky (cvičení), Ediční středisko VUT Brno, Brno, 1988, skriptum

[3] Mitrana, V.: Parsability approaches in CD Grammar Systems, technická zpráva

[4] Mihalache, V., Mitrana, V.: Deterministic Cooperating Distributed Grammar Systems, In: Turku Centre for Computer Science, TUCS Technical Report No 63, ISBN 951-650-886-3, ISSN 1239-1891

[5] Middendorf, M.: Supersequences, Runs and CD Grammar Systems, technická zpráva

[6] Fernau, H.: Regulated Grammars under Leftmost Derivation, technická zpráva, 1999

# A Kompletní analyzátor gramatiky 2

```
void S(int n) {
  insert(AC, 1);
  call();
  call();
}
void X(int n) { insert(A); }
void Y(int n) { insert(C); }
void A(int n) {
  if (n==0) {
    if (alex=='a') A(2);
    else
    if (alex=='d') A(3);
    else Error();
  } else
  if (n==2) {
    insert(``aXb'', 2);
    expect('a');
    call(); // navrat bod #1
    expect('b');
  } else
  if (n==3) {
    insert(``d'',3);
    expect('d');
  }
}
void C(int n) {
   if (n==2) {
      insert(cY, 2);
      expect('c');
      call();
   } else
   if (n==3) {
      insert(d);
      expect('d');
   }
}
```