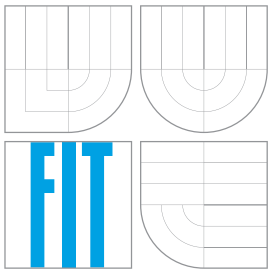
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VYUŽITÍ GRAMATIK S ROZPTÝLENÝM KONTEXTEM V PROGRAMOVACÍCH JAZYCÍCH

APPLICATIONS OF SCATTERED-CONTEXT GRAMMARS IN PROGRAMMING LANGUAGES

PROJEKT DO PŘEDMĚTU TEORIE PROGRAMOVACÍCH JAZYKŮ
TJD PROJECT

AUTOR PRÁCE — PETR HORÁČEK
AUTHOR

VEDOUCÍ PRÁCE — Prof. Ing. TOMÁŠ HRUŠKA, CSc.
SUPERVISOR

BRNO 2010

Obsah

Kapitola 1

Úvod

Cílem tohoto textu je zkoumat možnosti využití tzv. gramatik s rozptýleným kontextem (scattered-context grammars) v programovacích jazycích. V první části stručně představuje v současnosti bežně používaný popis syntaxe programovacích jazyků pomocí bezkontextových gramatik. Dále první kapitola nabízí pohled na jednu ze známých metod syntaktické analýzy, tzv. LL(1) parsing. Veškeré principy jsou ilustrovány na příkladu velmi jednoduchého programovacího jazyka.

Druhá část je již věnována samotným gramatikám s rozptýleným kontextem. Uvádíme zde základní definice a vlastnosti těchto gramatik, opět doplněné o stručné příklady. Hlavní část se pak zabývá možnostmi využití gramatik s rozptýleným kontextem v programovacích jazycích (zejména popis sémantických závislostí v rámci syntaxe jazyka), a také zkoumá, zda, resp. jak by bylo možné upravit zmíněnou metodu LL(1) parsingu tak, aby se dala použít pro gramatiky s rozptýleným kontextem.

V závěru jsou shrnuty dosažené výsledky.

První část tohoto textu vychází především z [1]. Definice a obecné vlastnosti gramatik s rozptýleným kontextem jsou převzaty z [3], stejně jako možnosti aplikace v lingvistice. Část věnovaná využití v programovacích jazycích je založena především na [4], částečně také [2].

Kapitola 2

Syntaktická analýza programovacích jazyků

2.1 Základní definice

2.1.1 Bezkontextová gramatika

Bezkontextová gramatika je čtveřice

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

kde

- V je konečná abeceda
- T je konečná množina *terminálních symbolů*
- P je konečná množina *pravidel* ve tvaru

  $$A \to x,$$

  kde $A \in V - T$, $x \in V^*$
- S je *startovní symbol*

2.1.2 Derivační krok

Nechť $G = (V, T, P, S)$ je bezkontextová gramatika. Nechť $u, v \in V^*$ a $p = A \to x \in P$. Pak říkáme, že *uAv přímo derivuje uxv* v G podle p. Píšeme

$$u \Rightarrow v \ [p]$$

Relaci $\Rightarrow^*$ (*derivuje*) definujeme jako reflexivní a tranzitivní uzávěr relace $\Rightarrow$.

2.1.3 Generovaný jazyk

Nechť $G = (V, T, P, S)$ je bezkontextová gramatika. Pak *jazyk generovaný gramatikou G* definujeme jako

$$L(G) = \{x \in T^* : S \Rightarrow^* x\}$$

2.2 Příklad programovacího jazyka

Pro účely tohoto textu byl vytvořen velice jednoduchý programovací jazyk, inspirovaný především známým jazykem Pascal. Syntaxi jazyka popisuje bezkontextová gramatika s následujícími pravidly (nonterminály jsou zapsány velkými písmeny, startovní symbol je S):

1	S	$\rightarrow$	$DECL\ PROG$
2	$DECL$	$\rightarrow$	$var\ IDLIST$
3	$DECL$	$\rightarrow$	ϵ
4	$IDLIST$	$\rightarrow$	$id\ IDNEXT,$
5	$IDNEXT$	$\rightarrow$	$,\ id\ IDNEXT$
6	$IDNEXT$	$\rightarrow$	ϵ
7	$PROG$	$\rightarrow$	$begin\ STATLIST$
8	$STATLIST$	$\rightarrow$	$STAT\ ;\ STATLIST$
9	$STATLIST$	$\rightarrow$	end
10	$STAT$	$\rightarrow$	$read\ id$
11	$STAT$	$\rightarrow$	$write\ id$
12	$STAT$	$\rightarrow$	$id\ ASSIGN$
13	$ASSIGN$	$\rightarrow$	$:=\ id$

Pozn.: terminální symbol *id* je do jisté míry speciální v tom smyslu, že na rozdíl od ostatních terminálů neodpovídá přímo zápisu v kódu programu, ale značí identifikátor, který bude v našem programovacím jazyce pro jednoduchost definován regulárním výrazem $a(ab)^*$.

Příkladem korektního programu v tomto jazyce (tzn. věty generované výše uvedenou gramatikou) je:

```
var a, ab
begin
  read a;
  ab := a;
  write ab;
end
```

Korektní je ovšem např. i následující zápis:

```
var a, ab
begin
  read aba;
  aa := a;
  write aabb;
end
```

Jak zde můžeme vidět, je definována pouze syntaxe jazyka, bez ohledu na jakékoliv sémantické závislosti (v tomto případě deklarace a definice proměnných). Kdybychom chtěli rozšířit definici syntaxe tak, aby v sobě zahrnovala i tyto závislosti, zjistíme, že jazyk již nebude bezkontextový. V praxi se tedy obvykle setkáme s oddělením syntaktické a sémantické

části popisu jazyka. Často je syntaxe popsána bezkontextovou gramatikou, a sémantické závislosti jsou zajištěny pomocí tzv. sémantických akcí. Běžným řešením problému deklarace a definice proměnných (resp. konstant, funkcí apod.) je zavedení tzv. tabulky symbolů, v níž jsou uchovávány informace o tom, které proměnné byly deklarovány, resp. definovány, případně i další informace o jejich použití.

2.3 LL(1) parsing

V současnosti existuje velké množství metod syntaktické analýzy, celou řadu z nich můžeme najít např. v [1] (zde zájemci naleznou nejen teorii, ale i mnoho rad a návodů pro praktickou implementaci). V tomto textu se zaměříme na jednu z nich, a to tzv. *LL(1) parsing*.

Jedná se o poměrně populární metodu syntaktické analýzy shora dolů (*top-down*). Název LL značí, že zpracováváme vstup zleva doprava, a provádíme nejlevější derivaci (tj. přepisujeme vždy nejlevější nonterminál). Velkou výhodou je, že můžeme vždy deterministicky určit, které pravidlo se má použít, ato pouze na základě současného nonterminálu a jediného *tokenu* na vstupu (tj. jednoho nejlevějšího symbolu nezpracované části vstupního řetězce). Lze zobecnit na LL(k), kde $k \geq 1$ značí, kolik symbolů můžeme použít, v tomto textu ale budeme uvažovat pouze LL(1).

Hlavní nevýhoda spočívá v tom, že je možné tuto metodu použít pouze pro určitý typ bezkontextových gramatik, tzv. LL(1) gramatiky, které jsou obecně slabší než bezkontextové (později si ukážeme, jak můžeme tuto třídu gramatik definovat). Často ale mohou tyto gramatiky pro naše účely postačovat, jak uvidíme i na příkladě.

2.3.1 Množina $First$

Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Pro každé $x \in (N \cup T)^*$ definujeme množinu $First(x)$ jako

$$First(x) = \{a : a \in T, x \Rightarrow^* ay; y \in (N \cup T)^*\}$$

Neformálně řečeno, množina $First(x)$ je množina všech terminálů, kterými začínají řetězce derivovatelné z větné formy x.

Tyto množiny můžeme pro všechny nonterminály vypočítat pomocí následujícího iteračního algoritmu:

1. Inicializujeme všechny množiny $First$ na prázdnou množinu.
2. Postupně procházíme všechna pravidla. Pokud pravá strana pravidla začíná terminálním symbolem, přidáme tento symbol do množiny $First$ levé strany pravidla. Pokud pravá strana začíná nonterminálním symbolem, přidáme všechny symboly z množiny $First$ tohoto symbolu do množiny $First$ levé strany pravidla. Formálněji to můžeme popsat takto:

   Pro každé pravidlo $A \to ay \in P$, kde $A \in N$, $a \in (N \cup T)$, $y \in (N \cup T)^*$:

   - Pokud $a \in T$, pak $First(A) := First(A) \cup \{a\}$.
   - Pokud $a \in N$, pak $First(A) := First(A) \cup First(a)$.
3. Opakujeme krok 2, dokud jsou aspoň do jedné množiny $First$ přidávány nové symboly.

Můžeme rozšířit definici množiny $First$ tak, že umožníme, aby kromě terminálních symbolů obsahovala také symbol ϵ. Pro všechny větné formy x takové, že $x \Rightarrow^* \epsilon$, přidáme ϵ do $First(x)$. Dále definujeme množinu $First(\epsilon) := \{\epsilon\}$.

Výpočet množin $First$ pro jednotlivé nonterminály pak bude probíhat podobně jako v předchozím případě. Musíme ovšem doplnit krok 2 následujícím způsobem:

Pro každé pravidlo $A \to ay \in P$, kde $A \in N$, $a \in (N \cup T)$, $y \in (N \cup T)^*$:

- Pokud $a \in T$, pak $First(A) := First(A) \cup \{a\}$.
- Pokud $a \in N$, pak $First(A) := First(A) \cup First(a)$.
- Pokud $a \in N$ a $\epsilon \in First(a)$, pak $First(A) := First(A) \cup First(y)$.

Neformálně řečeno, pokud symbol a lze přepsat na ϵ, můžeme jej „vynechat“, a tedy musíme vzít v úvahu i následující symbol. Názorněji tento princip ilustrujme příkladem.

Konstrukci množin $First$ pro výše uvedenou gramatiku našeho jednoduchého programovacího jazyka zachycuje následující tabulka (sloupce odpovídají jednotlivým iteracím algoritmu). V první iteraci jsou zatím všechny množiny $First$ prázdné, a přidáváme tedy pouze terminály (z pravidel, jejichž pravé strany začínají terminálem). Zajímavější je druhá iterace, kde je vidět využití ϵ. Do $First(S)$ jsme přidali var z $First(DECL)$, ale protože $First(DECL)$ obsahuje ϵ, musíme vzít v úvahu i následující symbol, tj. $PROG$. Přidáme tedy také $begin$ z $First(PROG)$. Ve třetí iteraci už nenastala v množinách žádná změna, a algoritmus tedy skončil.

	1	2	3
$First(S)$		$var\ begin$	$var\ begin$
$First(DECL)$	$var\ \epsilon$	$var\ \epsilon$	$var\ \epsilon$
$First(IDLIST)$	id	id	id
$First(IDNEXT)$	$,\ \epsilon$	$,\ \epsilon$	$,\ \epsilon$
$First(PROG)$	$begin$	$begin$	$begin$
$First(STATLIST)$	end	$end\ read\ write\ id$	$end\ read\ write\ id$
$First(STAT)$	$read\ write\ id$	$read\ write\ id$	$read\ write\ id$
$First(ASSIGN)$	$:=$	$:=$	$:=$

2.3.2 Množina $Follow$

Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Pro každé $A \in N$ definujeme množinu $Follow(A)$ jako

$$Follow(A) = \{a : a \in T, S \Rightarrow^* xAay, x, y \in (N \cup T)^*\} \cup \{\# : S \Rightarrow^* xA, x, y \in (N \cup T)^*\}$$

Množina $Follow(A)$ je tedy množina všech terminálů, které mohou následovat bezprostředně po A v nějaké větné formě v G. Symbol $\#$ je speciální symbol, který značí konec řetězce.

Množiny $Follow$ pro všechny nonterminály můžeme vypočítat podobně jako $First$, s využitím iteračního algoritmu:

1. Zkonstruujeme množiny $First$.
2. Inicializujeme všechny množiny $Follow$ na prázdnou množinu.

3. Postupně procházíme všechna pravidla. Pro každý nonterminál B obsažený v pravé straně pravidla (tj. $A \rightarrow xBy$) přidáme všechny symboly z $First(y)$ do $Follow(B)$. Navíc pokud y derivuje ϵ, přidáme také všechny symboly z $Follow(A)$ do $Follow(B)$. Formálněji:

   Pro každé pravidlo $A \rightarrow \alpha \in P$, kde $A \in N$, $\alpha \in (N \cup T)^*$:

   - Pro každé $B \in N, \alpha = xBy, x, y \in (N \cup T)^*$:
     - $Follow(B) := Follow(B) \cup First(y)$
     - Pokud $y \Rightarrow^* \epsilon$, pak $Follow(B) := Follow(B) \cup Follow(A)$.

4. Opakujeme krok 3, dokud jsou aspoň do jedné množiny $Follow$ přidávány nové symboly.

Příklad pro naši gramatiku vidíme v následující tabulce.

	1	2
$Follow(S)$	#	#
$Follow(DECL)$	*begin*	*begin*
$Follow(IDLIST)$	*begin*	*begin*
$Follow(IDNEXT)$	*begin*	*begin*
$Follow(PROG)$	#	#
$Follow(STATLIST)$	#	#
$Follow(STAT)$	;	;
$Follow(ASSIGN)$	;	;

2.3.3 Parse table

Pomocí množin $First$ a $Follow$ nyní můžeme zkonstruovat tabulku syntaktické analýzy (*parse table*). Pro každé pravidlo $p = A \rightarrow \alpha \in P$ přidáme p na pozice $[A, a]$ v tabulce pro všechny terminální symboly a takové, že:

$$(a \in First(\alpha)) \vee (a \in Follow(A) \wedge \epsilon \in First(\alpha))$$

.

Pro náš příklad tedy bude vypadat parse table následovně:

	var	,	*begin*	*id*	*read*	*write*	:=	;	*end*	#
S	1		1							
$DECL$	2		3							
$IDLIST$				4						
$IDNEXT$		5	6							
$PROG$			7							
$STATLIST$				8	8	8			9	
$STAT$				12	10	11				
$ASSIGN$							13			

Aby byl syntaktický analyzátor používající tuto tabulku deterministický, musí být na každé pozici nejvýše jedna položka. Semiformálně můžeme tedy definovat LL(1) gramatiku jako bezkontextovou gramatiku, jejíž tabulka syntaktické analýzy (sestavená dle výše uvedeného algortimu) neobsahuje na žádné pozici více jak jednu položku.

Je patrné, že pro gramatiku z našeho příkladu je tato podmínka splněna, a jedná se tedy o LL(1) gramatiku.

Kapitola 3

Gramatiky s rozptýleným kontextem a programovací jazyky

V této kapitole se stručně seznámíme s problematikou tzv. *gramatik s rozptýleným kontextem (scattered-context grammars, SCG)*. Kromě základních definic zmíníme také některé vlastnosti těchto gramatik – zejména se zaměříme na generativní sílu. Nastíníme možnosti praktického využití, především v programovacích jazycích. Nakonec budeme uvažovat, jak provádět syntaktickou analýzu. Kapitola vychází zejména z [3].

3.1 Základní definice a vlastnosti

3.1.1 Gramatika s rozptýleným kontextem

Gramatika s rozptýleným kontextem je čtveřice

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

kde

- V je konečná abeceda
- T je konečná množina *terminálních symbolů*
- P je konečná množina *pravidel* ve tvaru

  $$(A_1, \ldots, A_n) \to (x_1, \ldots, x_n),$$

  kde $A_1, \ldots, A_n \in V - T$, $x_1 \ldots, x_n \in V^*$
- S je *startovní symbol*

3.1.2 Derivační krok

Nechť $G = (V, T, P, S)$ je gramatika s rozptýleným kontextem. Pokud

$$\begin{aligned} p &= (A_1, \ldots, A_n) \to (x_1, \ldots, x_n) \\ u &= u_1 A_1 \ldots u_n A_n u_{n+1} \\ v &= u_1 x_1 \ldots u_n x_n u_{n+1} \end{aligned}$$

kde $p \in P$ a $u_i \in V^*$ pro $1 \leq i \leq n$, říkáme, že u *přímo derivuje* v v G podle p. Píšeme

$$u \Rightarrow v \ [p]$$

Relaci $\Rightarrow^*$ (*derivuje*) definujeme jako reflexivní a tranzitivní uzávěr relace $\Rightarrow$.

3.1.3 Generovaný jazyk

Nechť $G = (V, T, P, S)$ je gramatika s rozptýleným kontextem. Pak *jazyk generovaný gramatikou* G definujeme jako

$$L(G) = \{x \in T^* : S \Rightarrow^* x\}$$

3.1.4 Generativní síla

V teorii formálních jazyků můžeme zkoumat řadu vlastností gramatik a tříd jazyků, které definují. Jedná se např. o tzv. *uzávěrové vlastnosti* (tj. uzavřenost dané třídy jazyků vůči různým množinovým operacím) nebo o problémy *rozhodnutelnosti*. To je však nad rámec tohoto textu. Zájemci mohou najít bližší informace k tomuto tématu např. v [3].

Pro nás je nyní důležitá především *generativní síla* gramatik s rozptýleným kontextem. Jedná se totiž o jednu z hlavních výhod, které by využití těchto gramatik v programovacích jazycích mohlo přinést ve srovnání s klasicky používanými bezkontextovými gramatikami.

Uvažujme následující gramatiku s rozptýleným kontextem:

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

kde $P = \{$

$$\begin{array}{lrcl} 1 & (S) & \rightarrow & (ABC), \\ 2 & (A,\ B,\ C) & \rightarrow & (aA,\ bB,\ cC), \\ 3 & (A,\ B,\ C) & \rightarrow & (\epsilon,\ \epsilon,\ \epsilon) \end{array}$$

$\}$.

Jazyk generovaný gramatikou G je

$$L(G) = \{a^n b^n c^n : n \geq 0\},$$

což není bezkontextový jazyk. Je tedy zřejmé, že generativní síla gramatik s rozptýleným kontextem je vyšší než bezkontextových (důkaz, že také každý jazyk generovaný bezkontextovou gramatikou lze generovat gramatikou s rozptýleným kontextem, je triviální).

Dokonce ovšem platí, že třída jazyků generovaných gramatikami s rozptýleným kontextem je ekvivalentní třídě rekurzivně vyčíslitelných jazyků. Tento důkaz již je podstatně komplikovanější, najít jej můžete např. v [3].

3.2 Využití v programovacích jazycích

3.2.1 Deklarace a definice proměnných

Kapitola 3.1 v [4] nabízí jedno z možných rozšíření bezkontextového programovacího jazyka pomocí gramatiky s rozptýleným kontextem. Popsané rozšíření umožňuje zahrnout kontrolu deklarace a definice proměnných přímo do syntaxe jazyka.

Princip můžeme ilustrovat na jazyce popsaném v kapitole 2.2. Místo bezkontextové gramatiky uvažujme gramatiku s rozptýleným kontextem s následujícími pravidly.

$$
\begin{array}{rrcl}
1 & (S) & \rightarrow & (DECL\ PROG\ S') \\
\\
2 & (DECL) & \rightarrow & (var\ IDLIST) \\
3 & (DECL) & \rightarrow & (\epsilon) \\
4 & (IDLIST,\ S') & \rightarrow & (id\ IDNEXT,\ D) \\
5 & (IDNEXT,\ S') & \rightarrow & (,\ id\ IDNEXT,\ D) \\
6 & (IDNEXT) & \rightarrow & (\epsilon) \\
\\
7 & (PROG) & \rightarrow & (begin\ STATLIST) \\
8 & (STATLIST) & \rightarrow & (STAT\ ;\ STATLIST) \\
9 & (STATLIST) & \rightarrow & (end) \\
10 & (STAT,\ D) & \rightarrow & (read\ id,\ D\ L) \\
11 & (STAT,\ D) & \rightarrow & (write\ id,\ D\ R) \\
12 & (STAT,\ D) & \rightarrow & (id\ ASSIGN,\ D\ L) \\
13 & (ASSIGN,\ D) & \rightarrow & (:=\ id,\ D\ R)
\end{array}
$$

Derivace bude probíhat analogicky k původnímu případu, ale v okamžiku, kdy je vygenerována deklarace proměnné, zároveň dojde k přepsání S' na D. Tím se zachová informace o tom, že proměná id byla deklarována. Pokud se proměnná id vyskytne na levé straně přiřazovacího příkazu (resp. jako argument příkazu $read$), D se přepíše na $D\ L$ (proměnná byla definována). Pro pravou stranu (resp. $write$) dojde k přepisu D na $D\ R$.

Pokud proměnná id je použita, aniž by byla deklarována, derivace nemůže pokračovat (S' nebylo přepsáno na D a nelze jej přímo přepsat na $D\ L$, resp. $D\ R$). Naopak po úspěšném dokončení syntaktické analýzy navíc můžeme podle vygenerovaných symbolů zjistit další informace. Dostaneme řetězec v jednom z následujících tvarů:

1. D — proměnná byla deklarována, ale nebyla potom použita v žádném příkazu
2. DL^+ — proměnná byla deklarována, ale byla použita pouze na levé straně přiřazení
3. $DR(RL)^*$ — proměnná byla deklarována, ale její první použití bylo na pravé straně přiřazení (a tedy dříve, než byla definována)
4. $DL^+R(RL)^*$ — korektní použití proměnné

Zásadní nevýhodou výše uvedeného algoritmu je, že můžeme takto zpracovat pouze jedinou proměnnou. Východiskem, nabízeným v [4] je rozšíření na konečný počet proměnných tak, že pro každou deklarovanou proměnnou přidáme nový symbol $S'^{\cdots\prime}$. Pro syntaktickou analýzu pro tyto gramatiky autor navrhuje použití zásobníkového automatu s více zásobníky.

3.3 Další možnosti využití

3.3.1 Zpracování přirozeného jazyka

Závěrečná část [3] popisuje možné aplikace gramatik s rozptýleným kontextem v lingvistice. Konkrétně je zde využito gramatik s rozptýleným kontextem k popisu některých gramatických jevů anglického jazyka.

Autoři zde mimo jiné zavádí tzv. transformační gramatiky s rozptýleným kontextem, které nám např. umožňují převádět anglické věty na jiné anglické věty (které jsou opět

korektní). Princip spočívá v upravení definice gramatik s rozptýleným kontextem tak, že jejich derivace nebudou začínat z jediného startovního symbolu, ale z určitého jazyka. Jedním z příkladů je gramatika, která z věty obsahující dvojici neither-nor utvoří korektně její negaci. Např. větu „Neither Thomas nor his wife went to the party.“ („Tomáš ani jeho žena nešli na večírek.“) převede na „Both Thomas and his wife went to the party.“ („Tomáš i jeho žena šli na večírek.“) Je zřejmé, že slova neither a nor spolu souvisejí, ale mezi nimi se může vyskytovat obecně neomezený počet dalších slov, což komplikuje situaci, pokud bychom chtěli použít např. kontextové gramatiky.

Řešení pomocí gramatiky s rozptýleným kontextem je relativně jednoduché. Množina pravidel P vypadá takto:

$$\begin{aligned} P = \quad & \{\ (\text{<neither>}, \text{<nor>}) \rightarrow (\text{both, and})\ \} \\ \cup \quad & \{\ (\text{<}x\text{>}) \rightarrow (x) : x \in T - \{\text{neither, nor}\ \}\ \} \end{aligned}$$

Pro náš příklad pak transformace může probíhat následovně:

<neither> <thomas> <nor> <his> <wife> <went> <to> <the> <party>
$\Rightarrow$ both <thomas> and <his> <wife> <went> <to> <the> <party>
$\Rightarrow$ both thomas and <his> <wife> <went> <to> <the> <party>
$\Rightarrow$ both thomas and his <wife> <went> <to> <the> <party>
$\Rightarrow^5$ both thomas and his wife went to the party

Podrobný popis tohoto i mnoha dalších zajímavých příkladů, a také formální definice transformačních gramatik můžete najít v [3].

3.4 LL(1) parsing

Pro praktické použití je zásadní otázka, jak by se dala efektivně provádět syntaktická analýza podle gramatik s rozptýleným kontextem. V kapitole 2.3 jsme popsali jeden z populárních způsobů parsingu bezkontextových jazyků pomocí LL(1) gramatik. Zkusme se nyní zamyslet, zda bychom nemohli LL(1) parsing upravit, aby se dal použít i pro gramatiky s rozptýleným kontextem.

Zjednodušeně můžeme říci, že jedno pravidlo s rozptýleným kontextem je n-tice bezkontextových pravidel, která musejí být použita současně (resp. aplikována bezprostředně po sobě, v daném pořadí). Asi nejjednodušším způsobem, jak definici LL(1) gramatik zobecnit z bezkontextových na gramatiky s rozptýleným kontextem, je uvažovat každé bezkontextové pravidlo z každé n-tice samostatně. Pak můžeme zkonstruovat množiny $First$ a $Follow$ stejně jako v případě bezkontextových gramatik.

Pro gramatiku uvedenou v kapitole 3.1.4, která generuje jazyk $a^n b^n c^n$, získáme následující množinu bezkontextových pravidel:

$$\begin{array}{rlcl} 1 & S & \rightarrow & ABC \\ 2\text{a} & A & \rightarrow & aA \\ 2\text{b} & B & \rightarrow & bB \\ 2\text{c} & C & \rightarrow & cC \\ 3\text{a} & A & \rightarrow & \epsilon \\ 3\text{b} & B & \rightarrow & \epsilon \\ 3\text{c} & C & \rightarrow & \epsilon \end{array}$$

Konstrukcí množin $First$ a $Follow$ dle výše uvedeného algoritmu pak dostáváme:

$First(S)$	$a\ \epsilon$
$First(A)$	$a\ \epsilon$
$First(B)$	$b\ \epsilon$
$First(C)$	$c\ \epsilon$

$Follow(S)$	$\#$
$Follow(A)$	$b\ c\ \#$
$Follow(B)$	$c\ \#$
$Follow(C)$	$\#$

Tabulka syntaktické analýzy:

	a	b	c	$\#$
S	1			1
A	2a	3a	3a	3a
B		2b	3b	3b
C			2c	3c

LL(1) gramatiku s rozptýleným kontextem můžeme pak analogicky definovat jako gramatiku s rozptýleným kontextem, jejíž parse table obsahuje na každé pozici nejvýše jednu položku.

Na závěr ještě musíme parse table upravit tak, aby obsahovala původní pravidla s rozptýleným kontextem. V levém sloupci nebude ovšem levá strana pravidla, ale aktuální (nejlevější) nonterminál. Tabulka nám určuje, které pravidlo musíme použít pro daný nonterminál (nejlevější v aktuální větné formě) a daný symbol (nejlevější symbol dosud nezpracované části vstupního řetězce).

	a	b	c	$\#$
S	1			1
A	2			3

Pokud se ale na tímto problémem dále zamyslíme, můžeme dospět k závěru, že výše uvedená definice je zbytečně striktní. Např. gramatiku z kapitoly 3.2.1 bychom pak nemohli brát jako LL(1) (mj. kvůli pravidlům jako $D \rightarrow D\ L$ s levou rekurzí). Když ale uvážíme původní smysl LL(1) parsingu, tj. abychom mohli vždy rozhodnout, které pravidlo máme použít, pouze na základě současného nontermínálu a prvního symbolu nezpracované části vstupu, jsou pro nás důležité vždy jen první částí pravidel s rozptýleným kontextem (první bezkontextové pravidlo v n-tici). Omezíme-li se opět na nejlevější derivace, případné další části pravidla můžeme v tomto okamžiku zanedbat.

Podle této definice už gramatiku z kapitoly 3.2.1 můžeme chápat jako LL(1) gramatiku s rozptýleným kontextem. Konstrukce množin $First$ a $Follow$ bude pak probíhat zcela identicky jako v kapitole 2.3 a dostaneme opět totožný parse table:

	var	,	*begin*	*id*	*read*	*write*	:=	;	*end*	#
S	1		1							
$DECL$	2		3							
$IDLIST$				4						
$IDNEXT$		5	6							
$PROG$			7							
$STATLIST$				8	8	8			9	
$STAT$				12	10	11				
$ASSIGN$							13			

V tomto případě dokonce už není třeba žádná další úprava a tabulku můžeme přímo použít pro syntaktickou analýzu.

Kapitola 4

Závěr

V textu jsme se pokusili nalézt a zhodnotit možnosti aplikace gramatik s rozptýleným kontextem v programovacích jazycích. Z dosavadních výsledků můžeme dojít k závěru, že ačkoliv gramatiky s rozptýleným kontextem jsou velice silným nástrojem, v programovacích jazycích jejich uplatnění zřejmě nebude příliš velké. Drtivá většina současných běžně používaných programovacích jazyků je navržena tak, aby jejich syntaxe byla bezkontextová.

Zajímavější situace nastává, pokud se pokusíme v rámci syntaktické analýzy postihnout také některé sémantické závislosti v programovacích jazycích, jako je např. problém deklarace a definice proměnných. Zde už nám bezkontextové gramatiky stačit nebudou, a otevírá se tak prostor pro použití silnějších modelů. I tak se ovšem zatím zdá, že právě gramatiky s rozptýleným kontextem nebudou ideální cestou, nebo přinejmenším nepřináší dostatečně významné výhody v porovnání s existujícími metodami (zejména z praktického hlediska).

Podstatně větší uplatnění tak patrně gramatiky s rozptýleným kontextem mohou nalézt např. ve zpracování přirozených jazyků. V přirozených jazycích najdeme řadu kontextových závislostí, a to často takových, které by bylo poměrně obtížné postihnout i kontextovými gramatikami (závislosti mezi slovy, které nejsou bezprostředně vedle sebe). Navíc kontextové gramatiky nejsou obecně příliš vhodné pro využití v praxi (časová složitost syntaktické analýzy).

Významným zjištěním je bezpochyby fakt, že ačkoliv jsou gramatiky s rozptýleným kontextem obecně velmi silné (bez zavedení dalších omezení dokonce stejně jako gramatiky typu 0), syntaktická analýza by mohla být jednodušší, než v případě obecných gramatik, nebo dokonce kontextových. Přinejmenším pro určité typy gramatik s rozptýleným kontextem, jako v tomto textu navrhované LL(1), které jsou ovšem zřejmě stále silnější, než bezkontextové (viz příklady v tomto textu). Díky přímé návaznosti gramatik s rozptýleným kontextem na bezkontextové by mohlo být možné upravit některé další metody syntaktické analýzy, používané v případě bezkontextových gramatik.

Literatura

[1] Grune, D.; Jacobs, C. J. H.: *Parsing Techniques*. Springer, 2008, iSBN 978-0-387-20248-8.

[2] Křivka, Z.: *Využití řízených gramatik v programovacích jazycích*. projekt do předmětu TJD, 2006.

[3] Meduna, A.; Techet, J.: *Scattered Context Grammars and Their Applications*. WIT Press, 2009, iSBN 978-1-84564-426-0.

[4] Rychnovský, L.: *Parsing of Context-Sensitive Languages*. projekt do předmětu TJD, 2007.