Principy a podmínky LL analýzy

Thursday, May 30, 2013 8:42 AM

LL-gramatika

LL gramatika je jakákoliv gramatika, z níž se dá udělat rozkladová tabulka pro LL parser. **LL(k) parser** se kouká při parsování věty na následujících *k* tokenů, aby věděl, co dál. Pokud takový parser může být použit pro nějakou gramatiku, aniž by se musel použít backtracking, jedná se o **LL(k) gramatiku**.

Aby se ze vstupní gramatiky dala udělat LL(1) gramatika – eliminace levé rekurze, levá faktorizace (eliminace překrývajících se množin FIRST, tj. "rozsekání" pravidel se stejným začátkem pravé strany na několik menších, už jednoznačných)

př: STAT => if EXP then STAT | if EXP then STAT else → STAT => if EXP then STAT ElsePart; ElsePart => else STAT | e)

Podmínky

- Nesmí být přítomna levá rekurze.
- Nesmí dojít k first-follow (u neterminálu, který se přepisuje na "e") kolizi, first-first kolizi

Třídy jazyků LL(k)

L = Left to right -> vstupní text (soubor) čteme zleva doprava

L = Left parse -> vytváříme levý rozklad

K = při rozhodování mezi pravidly potřebujeme vidět nejvýše *k* znaků z nepřečtené části vstupu

Tzn.: **LL(k)** gramatika provádí deterministický rozbor *čtením textu z **L**eva doprava*, s *použitím **L**evé derivace* a *prohlédnutí **k** dalších symbolů vstupního textu*.

- gramatika je typu **LL(k)**, jestliže ji lze použít pro deterministickou syntaktickou analýzu metodou shora dolů (tj. vytváříme levý rozklad) a při rozhodování mezi pravidly potřebujeme znát nejvýše **k** symbolů ze vstupu.
- jazyk je typu **LL(k)**, pokud je generován některou **LL(k)** gramatikou

LL(0) gramatika

- lze určit správné pravidlo aniž bychom předem potřebovali vidět nějaký znak na vstupu
- každý neterminál musí mít jen jednu jedinou pravou stranu (jen jedno přepisovací pravidlo)
- neumožňuje rekurzi
- prostě jen určují, jestli sekvence patří do jazyka nebo ne, žádné rozhodování není potřeba.
- LL(0) gramatiky jsou nevhodné pro popis programovacích jazyků, protože zde není možná rekurze a pro každý neterminál existuje právě jedno pravidlo (důsledek faktu, že gramatika se nemůže rozhodnout podle následujícího vstupního symbolu), tudíž mohou generovat jen jazyk s jediným slovem.
  From <http://cs.wiki pedia.org/wiki/LL_syntaktick%C3%BD_analyz%C3%A1tor>

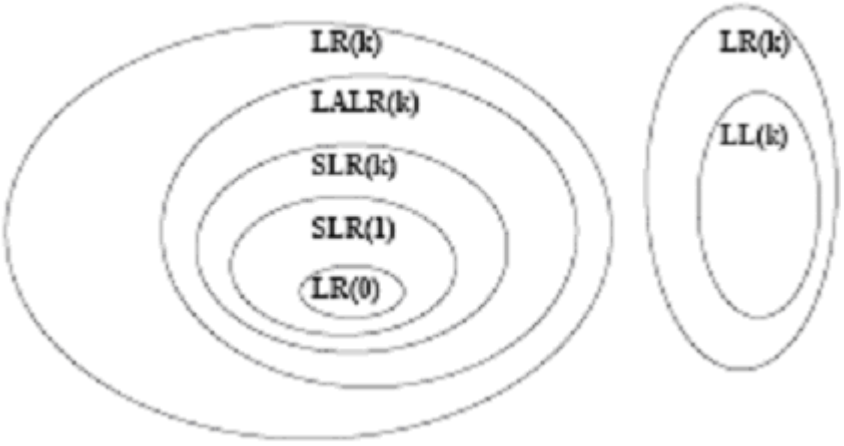
- Příklad:

```
G == id name lastname
id == [0-9]+
name == string
lastname == string
string == <unicode>+
```

LL(1) gramatika

- pro danou gramatiku *G* se vystačí při rozhodování o výběru pravidla pro expanzi s informací o dopředu prohlíženém řetězci délky 1 -> proto LL(1) je **gramatika silná**
- **jednoduchá LL (1) gramatika** je taková bezkontextová gramatika jestliže platí:
  - pravá strana každého pravidla začíná terminálním symbolem např. A -> aB
  - pokud mají 2 pravidla stejnou levou stranu, pak pravé strany začínají různými terminálními symboly např. A -> aB, A->bB
  - to znamená, že v každém políčku rozkladové tabulky bude právě jeden element
- **Obecná LL(1):**
  - gramatika nemá omezení, ale musí pro ni existovat rozkladová tabulka

Mohutnosti gramatik



Typy analýzy

- shora (top-down)
- sdola (bottom-up) (vyžadují LR gramatiku, takže se netýkají této otázky)

Analýza shora = analýza top-down

- Při hledání derivace začínáme počátečním symbolem a snažíme se dostat k hledanému slovu
- LL analýza: hledáme levou derivaci, vstupní slovo analyzujeme zleva
  - Přesně určuje volbu pravidel při analýze a umožňuje jednoznačný postup při odvození
  - Gramatika, která je jednoznačná a lze ji takto analyzovat: LL gramatika
  - Využívá se zásobníkový automat
- LL(k) označení gramatiky pro LL analýzu, číslo k určuje, kolik následujících symbolů na vstupu je nutné znát pro analýzu slova
- LL(1): nejpoužívanější gramatika, stačí znát jeden následující symbol
  - Jde vlastně o variantu rekurzivního sestupu bez backtrackingu
- LL(0): umožňuje jen jazyky s konečným počtem slov
- LL gramatiky s k>1 lze převést na LL gramatiky s k = 1
  - Existují přesné popisy, jak jednotlivá pravidla nahrazovat (přidávají se neterminály a pravidla se upravují, aby při analýze stačilo znát jeden další symbol)

LL parsery = parsery s analýzou top-down

LL parsery používají parsing **shora dolů**, zpracovávají vstup zleva doprava a konstruují nejlevější derivaci. Proto se také nazývá L (left-to-right) L(leftmost derivation). Občas se setkáváme s označením LL(k), kde k značí počet tokenů, které potřebujeme znát při rozhodování o průběhu další analýzy bez toho, aby bylo třeba používat backtracking (= prediktivní parser). Také se v této souvislosti používá pojem look-ahead. Prakticky do nedávné doby se tyto gramatiky příliš nepoužívaly, ovšem na počátku 90. let minulého století došlo ke změně přístupu.

Syntaktická analýza LL gramatik

Budeme se zabývat algoritmem syntaktické analýzy, který vytváří derivační strom analyzovaného řetězce směrem shora dolů. Základní princip syntaktické analýzy můžeme v tomto případě formulovat takto:

Je dána bezkontextová gramatika G= (N,T,P,S) a řetězec w = $a_1\ a_2 \ldots a_n$, který je větou z L(G). Pak existuje levá derivace

$S = \gamma_1 \Rightarrow \gamma_2 \Rightarrow \ldots \Rightarrow \gamma_n = w.$

Vzhledem k tomu, že derivace je levá, má každá větná forma γi tvar:

$\gamma i = a_1\ a_2 \ldots a_j\ A_i\ \beta_i,$

kde a_1, a_2 …, a_j jsou terminální symboly, A_i je neterminální symbol, β_i je řetězec terminálních a neterminálních symbolů. Přitom řetězec $a_1a_2 \ldots a_j$ je předponou věty w, $j \geq 0$.

Podmínky LL analýzy

Předpokládejme, že $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ jsou všechna pravidla v P s neterminálním symbolem A na levé straně. Pak základní problém syntaktické analýzy metodou shora dolů spočívá v nalezení toho pravidla $A \rightarrow \alpha_k$, jehož aplikací dostaneme z větné formy γ_i větnou formu $\gamma_i + 1$.

Pro výběr pravidla $A \rightarrow \alpha_k$, je možno použít:

1. informaci o dosavadním průběhu (historii) analýzy,
2. informaci o dosud nepřečtené části vstupního řetězce (dopředu prohlíženém řetězci omezené délky).

Pokud tyto informace vždy stačí k jednoznačnému výběru pravidla $A \rightarrow \alpha_k$, pak se gramatika G nazývá *LL gramatika*. Název je odvozen od toho, že při čtení vstupního řetězce zleva je vytvářen levý rozklad. Při syntaktické analýze LL gramatik jsou do zásobníku ukládány řetězce, které odpovídají levým větným formám nebo takovým jejich příponám, které vzniknou odejmutím předpony tvořené řetězcem terminálních symbolů.

Základními operacemi syntaktického analyzátoru pro LL gramatiky (LL analyzátoru) jsou:

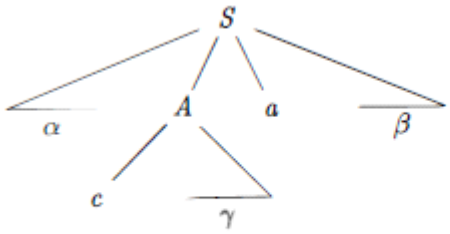
- **Expanze** – neterminální symbol na vrcholu zásobníku je nahrazen pravou stranou vybraného pravidla
- **Srovnání** – terminální symbol na vrcholu zásobníku se ze zásobníku vyloučí, jestliže je shodný se symbolem, který byl ze vstupního řetězce přečten.
- **Přijetí** – vstupní řetězec je přečten a zásobník je prázdný.
- **Chyba** – ve všech ostatních případech.

Pokud pro danou gramatiku G vystačíme při rozhodování o výběru pravidla pro expanzi s informací o dopředu prohlíženém řetězci délky nejvýše *k*, pak se gramatika G nazývá *silná LL(k) gramatika*. Při analýze silných LL(k) gramatik jsou do zásobníku ukládány přímo symboly gramatiky a syntaktický analyzátor je řízen rozkladovou tabulkou.

Funkce FIRST a FOLLOW

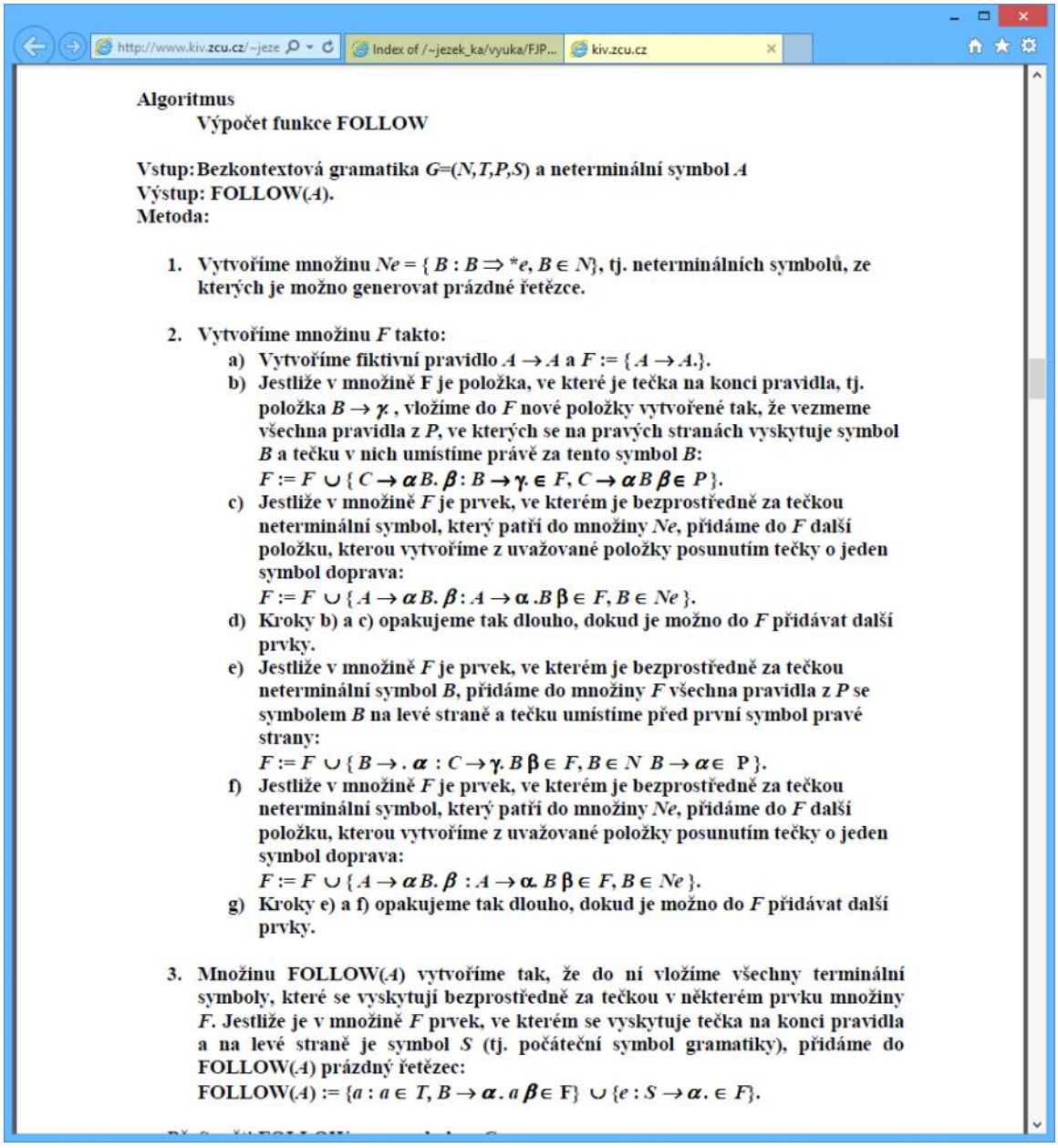
Konstrukce jak top-down, tak bottom-up parserů používá dvě funkce, FIRST a FOLLOW, spojené

s gramatikou G. Při parsování shora dolů nám FIRST a FOLLOW říkají, které přepisovací pravidlo uplatnit v závislosti na dalším vstupním symbolu. Během zotavení z chyby při panic módu mohou být množiny tokenů získané pomocí FOLLOW použity jako synchronizační tokeny.



Terminal c is in FIRST(A) and a is in FOLLOW(A)

Popis LL gramatik a LL analyzátoru



Algoritmus
Výpočet funkce FOLLOW

Vstup: Bezkontextová gramatika $G=(N,T,P,S)$ a neterminální symbol A
Výstup: FOLLOW(A).
Metoda:

1. Vytvoříme množinu $Ne = \{ B : B \Rightarrow^* e, B \in N\}$, tj. neterminálních symbolů, ze kterých je možno generovat prázdné řetězce.

2. Vytvoříme množinu F takto:
   a) Vytvoříme fiktivní pravidlo $A \rightarrow A$ a $F := \{ A \rightarrow A.\}$.
   b) Jestliže v množině F je položka, ve které je tečka na konci pravidla, tj. položka $B \rightarrow \gamma .$, vložíme do F nové položky vytvořené tak, že vezmeme všechna pravidla z P, ve kterých se na pravých stranách vyskytuje symbol B a tečku v nich umístíme právě za tento symbol B:
   $F := F \cup \{ C \rightarrow \alpha B. \beta : B \rightarrow \gamma. \in F, C \rightarrow \alpha B \beta \in P \}$.
   c) Jestliže v množině F je prvek, ve kterém je bezprostředně za tečkou neterminální symbol, který patří do množiny Ne, přidáme do F další položku, kterou vytvoříme z uvažované položky posunutím tečky o jeden symbol doprava:
   $F := F \cup \{ A \rightarrow \alpha B. \beta : A \rightarrow \alpha .B \beta \in F, B \in Ne \}$.
   d) Kroky b) a c) opakujeme tak dlouho, dokud je možno do F přidávat další prvky.
   e) Jestliže v množině F je prvek, ve kterém je bezprostředně za tečkou neterminální symbol B, přidáme do množiny F všechna pravidla z P se symbolem B na levé straně a tečku umístíme před první symbol pravé strany:
   $F := F \cup \{ B \rightarrow . \alpha : C \rightarrow \gamma. B \beta \in F, B \in N \; B \rightarrow \alpha \in P \}$.
   f) Jestliže v množině F je prvek, ve kterém je bezprostředně za tečkou neterminální symbol, který patří do množiny Ne, přidáme do F další položku, kterou vytvoříme z uvažované položky posunutím tečky o jeden symbol doprava:
   $F := F \cup \{ A \rightarrow \alpha B. \beta : A \rightarrow \alpha. B \beta \in F, B \in Ne \}$.
   g) Kroky e) a f) opakujeme tak dlouho, dokud je možno do F přidávat další prvky.

3. Množinu FOLLOW(A) vytvoříme tak, že do ní vložíme všechny terminální symboly, které se vyskytují bezprostředně za tečkou v některém prvku množiny F. Jestliže je v množině F prvek, ve kterém se vyskytuje tečka na konci pravidla a na levé straně je symbol S (tj. počáteční symbol gramatiky), přidáme do FOLLOW(A) prázdný řetězec:
   FOLLOW(A) := $\{a : a \in T, B \rightarrow \alpha . a \beta \in F\} \cup \{e : S \rightarrow \alpha . \in F\}$.

From <https://d.docs.live.net/e3534876709763a3/Dokumenty/ZCU/Statnice/Statnice.docx>

Řešení kolizí: http://www.kiv.zcu.cz/~lobaz/fjp/fjp11.html