Automaty a gramatiky

7

Roman Barták, KTIML

bartak@ktiml.mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Chomského hierarchie

gramatiky typu 0 (rekurzivně spočetné jazyky $\mathcal{L}_0$)
pravidla v obecné formě

gramatiky typu 1 (kontextové jazyky $\mathcal{L}_1$)
pouze pravidla ve tvaru $\alpha X\beta \rightarrow \alpha w\beta$,
$X \in V_N$, $\alpha,\beta \in (V_N \cup V_T)^*$, $w \in (V_N \cup V_T)^+$
jedinou výjimkou je pravidlo $S \rightarrow \lambda$, potom se ale S nevyskytuje na pravé straně žádného pravidla

gramatiky typu 2 (bezkontextové jazyky $\mathcal{L}_2$)
pouze pravidla ve tvaru $X \rightarrow w$, $X \in V_N$, $w \in (V_N \cup V_T)^*$

gramatiky typu 3 (regulární/pravě lineární jazyky $\mathcal{L}_3$)
pouze pravidla ve tvaru $X \rightarrow wY$, $X \rightarrow w$, $X,Y \in V_N$, $w \in V_T^*$

Bezkontextové gramatiky

pouze pravidla ve tvaru $X \rightarrow w$, $X \in V_N$, $w \in (V_N \cup V_T)^*$
velký praktický význam
- definování syntaxe vyšších programovacích jazyků
- konstrukce kompilátorů

Příklad:

```
<Program> → <Příkaz> | <Příkaz> <Program>
<Příkaz> → <IF-příkaz> | <WHILE-příkaz> | <Přiřazení>
<IF-příkaz> → if <Test> then <Příkaz> else <Příkaz>
<WHILE-příkaz> → while <Test> do <Příkaz>
<Přiřazení> → <Proměnná> := <Výraz>
```

Bude nás zajímat:
- jednoznačnost gramatiky (kvůli překladu)
- analýza pomocí zásobníkových automatů
- vlastnosti BKG obecně

Redukované bezkontextové gramatiky

Redukce (gramatiky) = vyřazení zbytečností

u konečných automatů:
- dosažitelné stavy, které nejsou ekvivalentní
- redukt určen jednoznačně

u bezkontextových gramatik:
- dosažitelné neterminály, které něco generují
- zde slabší význam (nemáme jednoznačnost)

Bezkontextová gramatika G (taková, že $L(G) \neq \emptyset$) se nazývá *redukovaná*, jestliže:
1) pro každý neterminál X existuje alespoň jedno terminální slovo w takové, že $X \Rightarrow^* w$
2) pro každý neterminál X různý od S existují slova u, v tak, že $S \Rightarrow^* uXv$ (dosažitelnost).

Redukce bezkontextových gramatik

Příklad:

S → aA | ab Zjevně stačí pravidlo
A → BC S → ab,
B → ba ostatní pravidla (neterminály) jsou
D → ab | λ zbytečná

Tvrzení: Ke každé bezkontextové gramatice G takové, že $L(G)\neq\emptyset$ lze sestrojit ekvivalentní redukovanou gramatiku.

Důkaz (idea):

1) vyhoď neterminály, které negenerují terminální slovo

2) vyhoď nedosažitelné neterminály

Poznámka: pořadí redukčních kroků nelze prohodit!

S → ab | A
A → BC
B → b

~~S → ab~~
~~B → b~~

Algoritmus redukce - krok 1

hledáme $V = \{X \mid X\in V_N, \exists w\in V_T^* \; X\Rightarrow^* w\}$

obvyklý postup (iterace po krocích):

$V_0 = V_T$

$V_{i+1} = V_i \cup \{X \mid X\in V_N, \exists w\in V_i^* \; (X\rightarrow w)\in P\}$

$V_0 \subseteq V_1 \subseteq \ldots \subseteq V_T \cup V_N$ + stabilizace ($\exists k \; V_k = V_{k+1} = \ldots$) + $V = V_k \cap V_N$

Zároveň víme, zda $L(G)\neq\emptyset$ ($L(G)\neq\emptyset \Leftrightarrow S\in V$)

Nyní z gramatiky odstraníme všechna pravidla obsahující na levé či pravé straně neterminál nepatřící do V.

a) máme splněn bod 1) definice redukované gramatiky

pro $X\in V$ víme: $\exists w\in V_T^* \; X \Rightarrow^* w$ + použitá pravidla nebyla odstraněna

b) získaná gramatika G‘ je ekvivalentní s původní gramatikou G

$L(G') \subseteq L(G)$ zřejmé, $L(G) \subseteq L(G')$ lze ukázat sporem

Příklad:

S → aA | ab, A → BC, B → ba, D → ab | λ

V = {S,B,D}

redukovaná pravidla: S → ab, B → ba, D → ab | λ

Algoritmus redukce - krok 2 (dosažitelnost)

hledáme $U = \{X \mid X\in V_N, S\Rightarrow^* uXv\}$

obvyklý postup (iterace po krocích):

$U_0 = \{S\}$

$U_{i+1} = U_i \cup \{X \mid X\in V_N, \exists Y\in U_i \; (Y\rightarrow uXv)\in P\}$

$U_0 \subseteq U_1 \subseteq \ldots \subseteq V_N$ + stabilizace ($\exists k \; U_k = U_{k+1} = \ldots$) + $U = U_k$

Podobně jako v kroku 1 odstraníme z gramatiky všechna pravidla obsahující na levé či pravé straně neterminál nepatřící do U.

a) máme splněn bod 2) definice redukované gramatiky

pro $X \in U$ víme: $S \Rightarrow^* uXv$ + použitá pravidla nebyla odstraněna

b) získaná gramatika G‘‘ je ekvivalentní s gramatikou G‘

$L(G'') \subseteq L(G')$ zřejmé , $L(G') \subseteq L(G'')$ lze ukázat sporem

c) platnost bodu 1) definice redukované gramatiky nebyla narušena

spojením $X \Rightarrow^* w$ a $S \Rightarrow^* uXv$

Příklad:

S → ab, B → ba, D → ab | λ U = {S}

finální redukovaná pravidla: S → ab

Bezkontextové gramatiky a derivace

pouze pravidla ve tvaru $X\rightarrow w$, $X\in V_N$, $w\in(V_N \cup V_T)^*$

Úmluva

neterminály = velká písmena
terminály = malá písmena
pravidla = X→ u | v | w | … (pro stejnou levou stranu)

Derivace

$w\Rightarrow z$, jestliže: $\exists x,y,v\in(V_N \cup V_T)^*$ tž. $w=xAy$, $z=xvy$ a $(A\rightarrow v)\in P$

G: S → aSX | λ, X → XbSb | c

S ⇒ aSX ⇒ aSXbSb ⇒ aSXbb ⇒ aXbb ⇒ acbb

S ⇒ aSX ⇒ aX ⇒ aXbSb ⇒ acbSb ⇒ acbb

S ⇒ aSX ⇒ aSXbSb ⇒ aSXbb ⇒ aScbb ⇒ acbb

Pozorování

- stejná délka derivací (počet pravidel) + použita stejná pravidla
- liší se pořadím aplikace pravidel
- přepis neterminálu neovlivňuje derivaci ve zbytku slova

Kanonické derivace

Zdá se zbytečné zabývat se derivacemi s různým pořadím pravidel.

Definice:

Levé přepsání $w \Rightarrow z$, jestliže se přepisuje nejlevější neterminál:
$\exists v,y \in (V_N \cup V_T)^* \; \exists x \in V_T^* \; \exists A \in V_N$ tž. $w=xAy$, $z=xvy$ a $(A \rightarrow v) \in P$

Levá derivace vzniká použitím pouze levých přepsání.

Pravé přepsání a *pravá derivace* se definuje obdobně (vždy se přepisuje nejpravější neterminál)

Lemma: Pro bezkontextové gramatiky platí:
$X \Rightarrow^* w$ právě tehdy, když existuje levá (pravá) derivace w z X

Důkaz:

stačí ukázat, že existence derivace implikuje existenci levé derivace

xAy ⇒ xuy, použitím $A \rightarrow u$

přepsání $A \rightarrow u$ neovlivní řetězce x a y ani jejich další přepisování

části x, A, y se přepisují nezávisle na sobě

můžeme preferovat aplikaci některých pravidel

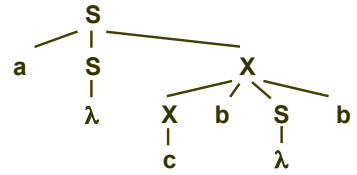
formální důkaz (indukcí dle délky derivace)

Derivační strom

Můžeme „výpočet“ zachytit jinak než sekvencí pravidel?

Příklad:

G: S → aSX |λ, X → XbSb | c



Definice:

Derivační strom je takový strom, že:

- každý vrchol je ohodnocen prvkem z $V_N \cup V_T \cup \{\lambda\}$
- kořen je ohodnocen S (počáteční neterminál)
- vnitřní vrcholy jsou ohodnoceny prvkem z V_N
- je-li A ohodnocení vrcholu a $u_1,\ldots,u_n$ jsou ohodnocení jeho potomků (bráno zleva doprava), potom $(A \rightarrow u_1,\ldots,u_n) \in P$
- je-li vrchol ohodnocen λ , potom je to list, který je jediným potomkem svého rodiče.

V derivačním stromu hraje roli jak stromové uspořádání dané hranami, tak uspořádání zleva doprava.

Derivace a derivační stromy

Říkáme, že *derivační strom dává slovo w*, jestliže *w* je slovo složené z ohodnocení listů (bráno zleva doprava).

Několik zřejmých tvrzení:

- S ⇒* w potom existuje derivační strom, který dává w (jednoznačně daný derivací)
- máme-li derivační strom, který dává w, potom S ⇒* w (derivace ale není určena jednoznačně)
- každý derivační strom jednoznačně určuje levou (a pravou) derivaci

Derivační strom zastupuje derivace slova získané „stejným způsobem“ (význam slova).

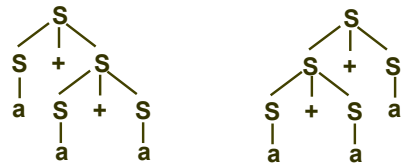
Je možné mít různé derivační stromy dávající stejné slovo (pro stejnou gramatiku)?

Jednoznačnost a víceznačnost BKG

Příklad:

S → S+S | a

slovo a+a+a



Definice:

- *Bezkontextová gramatika G je víceznačná* (nejednoznačná), jestliže ∃w∈L(G), které má dvě různé levé derivace.
- V ostatních případech *bezkontextová gramatika je jednoznačná*.
- *Bezkontextový jazyk L je jednoznačný*, jestliže existuje jednoznačná bezkontextová gramatika G tak, že L=L(G).
- *Bezkontextový jazyk L je (podstatně) nejednoznačný*, jestliže každá BKG G, taková že L=L(G) je nejednoznačná.

Příklad:

jazyk $\{a^i b^j c^k \mid i=j \vee j=k\}$ je podstatně nejednoznačný

pro slovo $a^i b^i c^i$ existují z principielních důvodů dva způsoby odvození

Od víceznačnosti k jednoznačnosti

Víceznačnost je potenciálním zdrojem potíží.
jedno slovo = více významů
U programovacích jazyků je víceznačnost nepřípustná!

Příklad 1:

S → S+S | a …víceznačná gramatika
S → a+S | a …ekvivalentní jednoznačná gramatika

Příklad 2:

S → if then S else S | if then S | λ
slovo „if then if then else“ má dva významy
„if then (if then else)“ nebo „if then (if then) else“
Řešení:

- syntaktická chyba (Algol 60)
- else patří k bližšímu if (preference pořadí pravidel)
- závorky begin-end (asi nejčistší řešení)

Jednoznačnost a kompilátory

E → E+E | E*E | (E) | a … nejednoznačné
E → E+E |T, T → T*T | F, F → (E) | a … řeší prioritu operací

Kompilace výrazu (zásobník na mezivýsledky+dva registry):

(1) E → E+T … pop r1; pop r2; add r1,r2; push r2
(2) E → T
(3) T → T*F … pop r1; pop r2; mul r1,r2; push r2
(4) T → F
(5) F → (E)
(6) F → a … push a

a+a*a, získáme postupnou aplikací pravidel 1,2,4,6,3,4,6,6

$\underline{E} \Rightarrow \underline{E}+T \Rightarrow \underline{T}+T \Rightarrow \underline{F}+T \Rightarrow a+\underline{T} \Rightarrow a+\underline{T}*F \Rightarrow a+\underline{F}*F \Rightarrow a+a*\underline{F} \Rightarrow a+a*a$

posloupnost obrátíme a vybere pouze pravidla generující kód
6,6,3,6,1
nyní pravidla nahradíme příslušným kódem

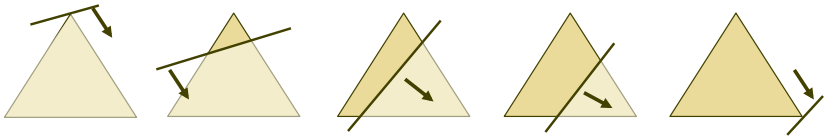
push a; push a; pop r1; pop r2; mul r1,r2; push r2; push a;
pop r1; pop r2; add r1,r2; push r2

Analýza shora bezkontextových jazyků

Jak k danému slovu a zvolené bezkontextové gramatice najdeme odpovídající derivační strom?

Analýza shora

- konstruujeme levou derivaci
- dosud vygenerované slovo kontrolujeme se vstupem



V každém kroku derivace můžeme slovo psát ve tvaru *uv*, kde

- *u* obsahuje pouze terminály (již přečtená část slova)
- *v* začíná neterminálem (zatím nehotová část)

Postup hledání derivace:

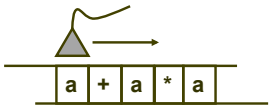
1) vezmi první neterminál *A* z *v* a nahraď ho *w*, dle pravidla *A→w*
2) vzniklé slovo *v* rozlož na *xy*, kde *x* obsahuje pouze terminály a *y* začíná neterminálem
3) zkontroluj *x* oproti vstupu a pokud je v pořádku, přidej *x* za *u*, polož *v* rovno *y* a opakuj od 1 dokud $v \neq \lambda$

Realizace analyzátoru

slovo generujeme na zásobník (LIFO struktura)
je-li vrchol zásobníku terminál, srovnáme ho se vstupem (čteme znak)
je-li vrchol zásobníku neterminál, nahradíme ho slovem dle pravidla
končíme, když je zásobník prázdný (musí být přečteno celé slovo)

Příklad:

(1) E → E+T
(2) E → T
(3) T → T*F
(4) T → F
(5) F → (E)
(6) F → a



zásobník	zbytek vstupu	pravidlo
E	a+a*a	
E+T	a+a*a	(1)
T+T	a+a*a	(2)
F+T	a+a*a	(4)
a+T	a+a*a	(6)
+T	+a*a	krácení
T	a*a	krácení
T*F	a*a	(3)
F*F	a*a	(4)
a*F	a*a	(6)
*F	*a	krácení
F	a	krácení
a	a	(6)
λ	λ	krácení