TI Referát

Filip Peterek

Listopad 2021

1 CNF

Startovací neterminál lze rozepsat na ε, poté se však startovací neterminál nesmí objevit na pravé straně žádného pravidla.

Jinak lze všechny neterminály rozepsat pouze na jeden terminál nebo dva jiné neterminály.

$S \to \varepsilon$
$A \to BC \mid CD \mid a \mid b$

2 Algoritmus Cocke-Younger-Kasami

Dynamické programování – přístup zdola nahoru (tabulation). Algoritmus CYK využívá vlastností CNF.

Přístup shora dolů by byl neefektivní - při rozpisu počátečního neterminálu bychom museli vyzkoušet vysoké množství kombinací. Přístup zdola nahoru nám umožní omezit množství kombinací, které musíme vyzkoušet, tím, že nejprve přepíšeme terminály na neterminály, a poté pouze hledáme pravidla, která dané neterminály mohly vytvořit.

Nejprve se všechny terminály přepíšou na odpovídající neterminály (jeden terminál může odpovídat více neterminálům). Tyto neterminály se zapíší do tabulky jako substringy délky 1.

Poté se prochází n-písmenné substringy ($n \in 2..length$) původního slova a hledají se pravidla, která odpovídají daným dvěma neterminálům (terminály byly eliminovány v prvním kroku a nahrazeny neterminály). Začíná se u substringů délky 2, délka substringů poté lineárně roste. Využívá se toho, že v CNF lze neterminál přepsat pouze na dva jiné neterminály, proto se substringy délky n rozdělí vždy jedině na dvě části ((1, n-1), (2, n-2), (3, n-3), ..., (n-1, 1)). Takto se nám substring rozdělí na dva kratší substringy – ovšem tyto kratší substringy již máme zpracované a zaznamenané v tabulce – proto dynamické programování (tabulation). Proto se podíváme pouze do tabulky, zda lze danou část substringu vygenerovat z některého z neterminálů gramatiky. Pokud ne, nelze ani původní substring vygenerovat pomocí současného dělení substringu a musíme zkoušet jiné rozdělení momentálně zkoušeného substringu. Takto postupujeme až dokud nevyzkoušíme substring délky n (tedy celé vstupní slovo).

Pokud se v množině neterminálů generujících původní slovo nachází startovací neterminál, můžeme říct, že $w \in \mathcal{L}(G)$.

3 Počet derivačních stromů

Pro zjištění počtu derivačních stromů stačí již pouze jednoduchá úprava algoritmu CYK.

3.1 Generování derivačních stromů

První možností, jak zjistit počet derivačních stromů, je vygenerovat všechny derivační stromy a poté je pouze spočítat. Tato možnost je flexibilnější, neboť umožňuje také získání samotných derivačních stromů, má ale vyšší paměťové i výkonové nároky, protože vyžaduje ukládání stromů v paměti a jejich následnou traverzaci.

Jádro algoritmu spočívá v ukládání derivačního podstromu pro každý neterminál. Pro každý neterminál si vytvoříme derivační strom, ve kterém si uložíme, ze kterých podstromů daný uzel vznikl. Do tabulky si poté místo neterminálů ukládáme přímo tyto podstromy.

Díky tomu, že je gramatika v CNF, bude strom vždy binární.

```
case class ParseNode(
  nt: NonTerminal,
  subnodes: (ParseNode, ParseNode)
)
```

Tímto způsobem můžeme vygenerovat více stromů pro jeden neterminál, pokud by daný neterminál šlo vytvořit pomocí více pravidel. Alternativně můžeme do derivačního stromu uložit seznam všech možných párů podstromů.

```
case class ParseNode(
  nt: NonTerminal,
  subnodes: Seq[(ParseNode, ParseNode)]
)
```

Toto řešení je ekvivalentní, musíme však počítat s tím, že sekvence subnodes obsahuje sadu nezávislých dvojic pravidel.

Aplikací algoritmu CYK takto získáme podobnou tabulku. Zda $w \in \mathcal{L}(G)$ zjistíme podle toho, že se na vrcholu tabulky nachází strom, jehož kořenem je počáteční neterminál.

Počty derivačních stromů poté získáme jednoduše – pro první případ stačí spočítat počet stromů s počátečním neterminálem na vrcholu tabulky.

Pro druhý případ stačí rekurzivně projít strom, jehož kořenem je počáteční neterminál, pro každou dvojici vynásobit počet derivačních podstromů pro levý a pravý neterminál (protože podstromy ve dvojici na sobě závisí), a sečíst počty derivačních stromů pro všechny dvojice (protože jednotlivé dvojice na sobě nezávisí).

3.2 Počítání derivačních stromů bez jejich generování

Pokud bychom nechtěli derivační stromy generovat, stačí nám místo derivačního stromu pro konkrétní neterminál uložit dvojici obsahující daný neterminál a počet pravidel, kolik daný neterminál na konkrétním místě dokáže vygenerovat (protože negenerujeme slova, ale jdeme opačným směrem). Aplikujeme podobný algoritmus výpočtu množství derivačních stromů, jako při předchozí možnosti, nyní ovšem množství derivačních stromů počítáme již při aplikaci algoritmu CYK. Počty derivačních stromů v rámci jedné dvojice násobíme, počty pro jednotlivé dvojice v rámci sekvence dvojic sčítáme. Pokud poté na vrcholu tabulky získáme počáteční neterminál, celkový počet derivačních stromů slova w v gramatice G je roven číslu ve dvojici s tímto neterminálem.