Transformace bezkontextových gramatik

V tomto cvičení se budeme zabývat základními transformacemi bezkontextových gramatik - BKG (též CFG, context-free grammar). Popsané postupy a myšlenky jsou jednoduché a vcelku přímočaré, leč začátečník se může na chvíli v záplavě písmenek ztratit. Stane-li se vám to, nezoufejte. Zkuste na věc přijít selským rozumem a uvidíte, že na tom nic není :).

Nechť G je bezkontextová gramatika daná čtveřicí (N, T, P, S), kde N je množina neterminálních symbolů, T množina terminálních symbolů, P přepisovací pravidla a S startovací symbol. Připomeňme si, že gramatika je bezkontextová, tvoří-li levou stranu všech přepisovacích pravidel právě jeden neterminální symbol, tj. pravidla jsou ve tvaru `X --> w`, kde X je neterminální symbol a ***w*** řetězec složený z terminálních i neterminálních symbolů.

Gramatika generuje nějaký jazyk, tj. množinu řetězců terminálních symbolů. Tato množina může být konečná, nekonečná nebo i prázdná. Postupně se budeme zabývat těmito otázkami: Je generovaný jazyk prázdný? Jsou v gramatice takové symboly, které by v ní být nemusely? Dá se něco udělat s přepisovacími pravidly, aby gramatika vypadala "lépe"?

Ukázka algoritmů

Minulý rok vznikly v rámci DP demonstrace pro uváděné algortimy, můžete se na ně podívat v dočasném úložišti, na převodu do portálu se zatím pracuje.

Nedostupné symboly

Začneme-li vytvářet ze startovacího symbolu S gramatiky G větné formy, může se stát, že k některým symbolům (terminálním či neterminálním) vůbec nedojdeme. Takové symboly nazveme nedostupné. Základem algoritmu k jejich vyhledání je přímočarý postup, kterým "prolézáme" pravidla a sespisujeme symboly, na které jsme narazili.

Můžete se podívat na demonstraci na webu.

Algoritmus

```
1) V0 = {S}, i=1
2) Vi = {X: A --> a X b je přepisovací pravidlo, a a b jsou řetězce symbolů, A náleží Vi-1} + Vi-1
3) Je-li Vi != Vi-1, pak i++ a jdi na (2), jinak
   VD = Vi (dostupné symboly)
   VN = N + T - VD (nedostupné symboly)
```

Gramatiku bez nedostupných symbolů zapíšeme snadno - budeme psát jen ta přepisovací pravidla, ve kterých figurují dostupné symboly. Zároveň upravíme i množiny N a T.

Příklady

1. G = (N, T, P, S), N = {S, A, B}, T = {a, b}, P:

   ```
   S --> aB | bB
   A --> a | BA
   B --> aS | bB
   ```

   Postupujeme podle algoritmu:

   ```
   V0 = {S}
   V1 = {S, B, a, b}
   V2 = {S, B, a, b} = V1, tj. VD = {S, B, a, b}
   ```

   Potom G' = {N', T', P', S}, kde N' = {S, B}, T' = {a, b} a P:

   ```
   S --> aB | bB
   B --> aS | bB
   ```

2. G = (N, T, P, S), N = {S, A, B, C}, T = {a, b}, P:

   ```
   S --> A | aB
   A --> a | BA
   B --> bB | e
   C --> ABC | S
   ```

   Postupujeme podle algoritmu:

   ```
   V0 = {S}
   V1 = {S, A, B, a}
   V2 = {S, A, B, a, b}
   V3 = V2, tj. VD = {S, A, B, a, b}
   ```

   Potom G' = {N', T', P', S}, kde N' = {S, A, B}, T' = {a, b} a P:

   ```
   S --> A | aB
   A --> a | BA
   B --> bB | e
   ```

Je jazyk prázdný?

Díky předchozímu textu umíme vyloučit z gramatiky takové symboly, které v ní nejsou evidentně užitečné. Za chvíli si ukážeme, že existuje ještě jeden typ takových symbolů. K tomu budeme potřebovat vědět, zda jazyk generovaný gramatikou G není prázdný, respektive budeme potřebovat algoritmus, který to zjistí.

Jazyk L(G) je množina řetězců terminálních symbolů. Gramatika může mít ale takovou vlastnost, že při derivacích (přepisování) nebudou ubývat neterminální symboly. Nyní tedy chceme zjistit, zda z gramatiky G vůbec lze takový řetězec generovat.

Můžete se podívat na demonstraci na webu.

Algoritmus:

```
1) N0 = {}, i = 1
```

```
2) Ni = {A: A --> a, a je řetězec složený ze symbolů Ni-1 a T} + Ni-1
3) Je-li Ni != Ni-1, i++ a jdi na (2), jinak
   Nt = Ni
4) Je-li S prvkem Nt, je L(G) neprázdný, v opačném případě je prázdný.
```

Základní myšlenka je prostá. Pravidla bývají rekurzivní. Aby se mohl generovat řetězec terminálních symbolů, musí se rekurze někde zastavit. Nejprve najdeme takové neterminální symboly, u kterých se rekurze může zastavit díky vhodnému přepisovacímu pravidlu (symbol se přepisuje na prázdný řetězec nebo na řetězec terminálních symbolů). Takovým symbolům budeme říkat "dobré". Má-li nějaký jiný neterminální symbol X na pravé straně přepisovacího pravidla řetězec z terminálních symbolů a "dobrých" neterminálních symbolů, je i X "dobrý". A tak dále. Je-li "dobrý" i startovací symbol, je celá gramatika "dobrá", tj. generuje neprázdný jazyk. Jasné?

Příklady

3. gramatika G z příkladu 1 bez nedostupných symbolů, tj. G = (N, T, P, S), N = {S, B}, T = {a, b}, P:

   ```
   S --> aB | bS
   B --> aS | bB
   ```

   Postupujeme podle algoritmu:

   ```
   N0 = {}
   N1 = N0, protože P neobsahuje žádné "dobré" pravidlo
   ```

   Jazyk L(G) je tedy prázdný.

4. G = (N, T, P, S), N = {S, A, B}, T = {a, b}, P:

   ```
   S --> A | aB
   A --> a | BA
   B --> bB | e
   ```

   Postupujeme podle algoritmu:

   ```
   N0 = {}
   N1 = {A, B}
   N2 = {S, A, B} = Nt
   ```

   Startovací symbol je "dobrý", tj. náleží do množiny N_t, tj. L(G) je neprázdný.

5. G = (N, T, P, S), N = {S, A, B, C, D}, T = {a, b}, P:

   ```
   S --> aAB | A
   A --> bC | D
   B --> A | b
   C --> S
   ```

   Připadá-li vám, že je v zadání chyba, neboť pravidla nedefinují přepisovací pravidla pro symbol D, dáváte pozor. Ale ne úplně, protože není nikde řečeno, že se všechny neterminální symboly musí vyskytovat na levé straně! Asi cítíte, že symbol D je v této gramatice zbytečný, tím se ale budeme zabývat až za chvíli. Nyní jen zjistíme, zda je L(G) prázdný nebo ne.

   ```
   N0 = {}
   N1 = {B}
   N2 = {B} = N1 = Nt
   ```

   Nejenže gramatika obsahuje zbytečný symbol, gramatika je nějak zbytečná celá, protože generuje prázdný jazyk.

Zbytečné symboly

Víme, že některé symboly gramatiky mohou být nedostupné. Ty jsou zbytečné automaticky. V posledním příkladu jsme viděli, že symbol D byl také zbytečný, protože pro něj neexistovalo přepisovací pravidlo. Pokud si celý problém formalizujeme, zjistíme, že přepisování probíhá asi takto:

$S \rightarrow^* a\ X\ b \rightarrow^* a\ c\ b$

kde *a*, *b* a *c* jsou řetězce terminálních symbolů, X je neterminální symbol a S startovací symbol. Šipka s hvězdičkou představuje přepsání, které může být sérií základních derivací. Pokud pro symbol X neexistuje takové přepsání, je symbol X zbytečný. Jinými slovy, jakmile se ve větné formě objeví X, nikdy už z ní nevyloučím neterminální symboly.

Odstranění zbytečných symbolů probíhá ve dvou krocích. Nejprve najdeme množinu N_t. Nová gramatika bude obsahovat jen tyto "dobré" neterminální symboly. Z nich je ale zapotřebí vyházet všechny nedostupné symboly. Snadné, že?

Příklady

6. G = (N, T, P, S), N = {S, A, B}, T = {a, b}, P:

   ```
   S --> ab | AB
   A --> AB
   B --> b
   ```

   Nejprve najdeme N_t:

   ```
   N0 = {}
   N1 = {S, B}
   N2 = N1 = Nt
   ```

   Pomocná gramatika je:

   ```
   S --> ab
   B --> b
   ```

   Vyloučíme nedostupné symboly:

   ```
   V0 = {S}
   V1 = {S, a, b} = VD
   ```

Gramatika bez zbytečných symbolů obsahuje jen jedno pravidlo, a to:

```
S --> ab
```

7. G = (N, T, P, S), N = {S, A, B, C, D}, T = {a, b, c}, P:

```
S --> aB | bD
A --> aB | bCD
B --> ABCD
C --> cC | A
D --> aD | e
```

Najdeme N_t:

```
N0 = {}
N1 = {D}
N2 = {S, D}
N3 = N2 = Nt
```

Nová gramatika má dvě pravidla (všechny symboly jsou dostupné):

```
S --> bD
D --> aD | e
```

Vyloučení pravidla

V tomto okamžiku umíme z gramatiky vyloučit všechny nepotřebné symboly. Někdy je také vhodné upravit či vyloučit nějaká pravidla. Princip je prostý: místo toho, aby se přepisování provedlo při derivaci větné formy, provedeme ho už v zápisu gramatiky. Potřebujeme-li pozměnit pravidlo `A --> aBb` a všechna pravidla pro B jsou `B -->` $c_1, c_2, \ldots, c_n$, přepíšeme pravidlo pro A na `A -->` $ac_1b, ac_2b, \ldots, ac_nb$ (*a, b, c* jsou řetězce terminálních symbolů)

Speciálními pravidly, kterých se někdy potřebujeme zbavit, jsou tzv. e-pravidla, tj. přepsání neterminálního symbolu na prázdný řetězec. Gramatikou bez e-pravidel pak budeme rozumět takovou gramatiku, ve které nejsou žádná e-pravidla, až na jednu výjimku. Tou je pravidlo `S --> e`, které do gramatiky zahrneme v případě, že gramatika generuje prázdný řetězec (S je startovací symbol). Potom se ovšem nesmí symbol S objevit na žádné z pravých stran přepisovacích pravidel.

Algoritmus:

1) Způsobem podobným jako při hledání N_t najdeme N_e, množinu symbolů, které se dají přepsat na e.

2) Pravidlo `A -->` a_0 `B`$_1$ a_1 `B`$_2$ a_2 `...` `B`$_n$ a_n nahradíme pravidly `A -->` a_0 `X`$_1$ a_1 `X`$_2$ a_2 `...` `X`$_n$ a_n, kde X_i je buď e nebo B_i (vystřídáme všech 2^i možností). Pravidlo typu `A --> e` pochopitelně zapisovat nebudeme, protože takových pravidel se snažíme zbavit.

3) Patří-li startovací symbol S do množiny N_e, zařadíme do gramatiky pravidlo `S' --> S | e`. Novým startovacím symbolem pak bude S'.

Příklady

8. G = (N, T, P, S), N = {S, A}, T = {a, b}, P:

```
S --> aSbS | aSbA | A
A --> aA | e
```

Najdeme množinu N_e:

```
N0 = {}
N1 = {A}       (protože A se dá přepsat na e)
N2 = {A, S}    (protože S se dá přepsat na A)
Ne = N2
```

Nyní budeme postupně vynechávat v pravidlech ty symboly, které se dají přepsat na e:

```
S --> aSbS | aSb | abS | ab |
      aSbA | abA |
      A
A --> aA | a
S' --> S | e   (protože S patří do Ne)
```

9. G = (N, T, P, S), N = {S, A, B}, T = {a, b}, P:

```
S --> aB | bA
A --> aA | B | e
B --> bB | A | e
```

Najdeme množinu N_e:

```
N0 = {}
N1 = {A, B}
N2 = N1 = Ne
```

Po přepisu:

```
S --> aB | bA | a | b
A --> aA | a | B
B --> bB | b | A
```

Jednoduchá pravidla

Jednoduché pravidlo ja takové, při kterém se neterminální symbol přepisuje na jiný neterminální symbol, tedy pravidlo typu `A --> B`. Sama o sobě by taková pravidla v gramatice nevadila, i když se dá diskutovat nad tím, zda jsou skutečně nutná. Hlavní motivací jejich odstranění je vedlejší jev, a to cykly. Na gramatice v posledním příkladu můžeme takový cyklus vidět: A se přepisuje na B a to opět na A. Odstraněním jednoduchých pravidel tedy odstraníme i cykly, které jsou nežádoucí při syntaktické analýze.

Nejprve pro každý neterminální symbol zjistíme, na jaké neterminální symboly se může přepsat pomocí jednoduchých pravidel. To se dá

zjistit postupem podobným jako při hledání dostupných symbolů. Pro neterminální symbol A zařadíme do množiny N_A jak symbol A, tak všechny "jednoduše dostupné" symboly.

V dalším kroku již píšeme nová pravidla. Pro neterminální symbol A napíšeme vechna pravidla A --> *a*, kde *a* jsou ne-jednoduchá pravidla pro všechny symboly z N_A.

Příklady

10. G = (N, T, P, E), N = {E, T, F}, T = {+, *, (,), i}, P:

    ```
    E --> E + T | T
    T --> T * F | F
    F --> (E) | i
    ```

    V této gramatice se sice cykly nevyskytují, ale z cvičných důvodů jednoduchá pravidla odstraníme:

    ```
    NE = {E, T, F}
    NT = {T, F}
    NF = {F}
    ```

    Nová gramatika je tedy:

    ```
    E --> E + T | T * F | (E) | i
    T --> T * F | (E) | i
    F --> (E) | i
    ```

11. G = (N, T, P, S), N = {A, B, S}, T = {a, b}, P:

    ```
    S --> aB | bA
    A --> aA | a | B
    B --> bB | b | A
    ```

    Sestavíme množiny $N_?$:

    ```
    NS = {S}              (zajímají nás jen jednoduchá pravidla!)
    NA = {A, B}
    NB = {A, B}
    ```

    Nová gramatika je:

    ```
    S --> aB | bA
    A --> a | aA | b | bB
    B --> a | aA | b | bB
    ```

Vlastní gramatika je gramatika bez zbytečných symbolů, bez e-pravidel a bez cyklů.

Odstranění levé rekurze

Pravidlo je levorekurzivní, je-li ve tvaru A --> A.... Gramatiky s levorekurzivními pravidly jsou nevhodné pro některé typy syntaktické analýzy. Jsou-li všechna pravidla pro neterminální symbol A zapsána jako

```
A --> Aa1 | Aa2 | ... | Aam |
      b1 | b2 | ... | bn
```

kde a_i a b_i jsou řetězce terminálních a neterminálních symbolů a žádné b_i nezačíná symbolem A, můžeme odstranit přímou levou rekurzi takto:

```
A  --> b1 | b2 | ... | bn |
       b1A' | b2A' | ... | bnA'
A' --> a1 | a2 | ... | am |
       a1A' | a2A' | ... | amA'
```

nebo pomocí e-pravidel:

```
A  --> b1A' | b2A' | ... | bnA'
A' --> e | a1A' | a2A' | ... | amA'
```

Sice jsme se učili e-pravidla odstraňovat, ale někdy jsou menší zlo.

Nepřímá levá rekurze je například toto: A --> B..., B --> A.... Její odstranění funguje podobně jako u přímé rekurze. Je velmi vhodné si do odstraňování zavést nějaký systém, jinak je slušná šance, že se v záplavě písmenek člověk ztratí!

Jeden ze systémů je následující. Označíme si neterminální symboly jako N_1, N_2, ..., N_n. Jsou-li v gramatice pravidla typu N_i --> N_j..., budeme se snažit, aby pokud možno i < j , tj. aby se přepisování neodkazovalo na předcházející symboly. Není to nutné, ale může to ušetřit práci. Pro pravidlo N_i --> ... nejprve zajistíme, aby se neodkazovalo na předcházející symboly. I v tom je třeba zavést pořádek, proto nejdříve odstraníme odkaz na N_1, pak na N_2 atd. Až na závěr si necháme odstranění přímé levé rekurze N_i --> N_i. Toto "čištění" pravidel provedeme postupně pro N_1, N_2 atd.

Příklady

12. G = (N, T, P, E), N = {E, T, F}, T = {+, *, (,), i}, P:

    ```
    E --> E + T | T
    T --> T * F | F
    F --> (E) | i
    ```

    Vidíme, že zde nepřímá levá rekurze není, budeme mít méně práce. Nejprve bez e-pravidel:

    ```
    E  --> T | TE'
    E' --> + T | + TE'
    T  --> F | FT'
    T' --> * F | * FT'
    F  --> (E) | i
    ```

V dalším cvičení zjistíme, že pro syntaktickou analýzu je vhodnější zavedení e-pravidel:

```
E  --> TE'
E' --> e | + TE'
T  --> FT'
T' --> e | * FT'
F  --> (E) | i
```

13. G = (N, T, P, A), N = {A, B, C}, T = {a, b}, P:

```
A --> AB | C | Bb
B --> Ab | Ca
C --> Aa | b
```

Zde se nepřímá levá rekurze vyskytuje. Při uspořádání symbolů A-B-C se B i C odkazují na předešlý symbol, A. V uspořádání B-C-A se na předešlý symbol odkazuje jen A. Použijeme tedy pořadí B-C-A.

```
B  --> Ab | Ca      (první pravidlo se nemůže odkazovat na předešlé symboly,
                     přímá levá rekurze zde není)
C  --> Aa | b       (neodkazuje se na B, přímá levá rekurze zde není)

   A --> AB | C | Bb                        (původní pravidla)
   A --> AB | C | Abb | Cab                 (odstranili jsme odkaz na B)
   A --> AB | Aa | b | Abb | Aaab | bab     (odstranili jsme odkaz na C)

A  --> b | bab | bA' | babA'                (nyní už píšeme definitivní pravidla)
A' --> B | a | bb | aab | BA' | aA' | bbA' | aabA'
```

S e-pravidly:

```
B  --> Ab | Ca
C  --> Aa | b

   A --> AB | C | Bb                        (původní pravidla)
   A --> AB | C | Abb | Cab                 (odstranili jsme odkaz na B)
   A --> AB | Aa | b | Abb | Aaab | bab     (odstranili jsme odkaz na C)

A  --> bA' | babA'                         (definitivní pravidla)
A' --> e | BA' | aA' | bbA' | aabA'
```

Příklady na domácí procvičení (vyřešte nejméně 3)

1. Převeďte G = (N, T, P, S) na gramatiku bez zbytečných symbolů:

```
S --> A | B
A --> aB | bS | b
B --> AB | Ba
C --> AS | b
```

2. Odstraňte u G=(N, T, P, S) e-pravidla:

```
S --> ABC
A --> BB | e
B --> CC | a
C --> AA | b
```

3. Odstraňte u G=(N, T, P, S') jednoduchá pravidla:

```
S' --> S | e
S  --> A | B
A  --> C | D
B  --> D | E
C  --> S | a
D  --> S | b
E  --> S | c
```

4. Odstraňte u G=(N, T, P, A) levou rekurzi bez použití e-pravidel:

```
A --> Ba | c
B --> CA | bb
C --> Ab | Bb
```

5. Odstraňte u G=(N, T, P, S) levou rekurzi s použitím e-pravidel:

```
S --> AB
A --> BS | b
B --> SA | a
```

Poslední změna: 19.11.2012