Normal Forms

Suradet Jitprapaikulsarn

First semester 2005

We can transform any context-free grammar (CFG) to Greibach normal forms which ensure that left recursion, direct or indirect, cannot occur. The steps are as follows:

1. Eliminating the recursion of the start symbol.
2. Eliminating the null varialbes.
3. Eliminating the chain rules.
4. Eliminating the useless symbols.
5. Transform the grammar to a Chomsky normal form.
6. Transfrom the modifed grammar to a Greibach normal form.

The resulting grammars (in a Greibach normal form) ensure that the top-down parsers will terminate.

1 Eliminating a recursion of the start symbol

The start symbol has a recursion if there is a derivation of the form $S \stackrel{*}{\Rightarrow} uSv$. We can eliminate this recursive derivation by

1. designating a new start symbol S'.
2. adding a new rule $S' \rightarrow S$ to the rules of the grammar.

Example 1 $G = (N, T, P, S)$ where
$N = \{S, A, B, C\}$
$T = \{a, b, c, \lambda\}$
$P : S \rightarrow AB \mid AC$
$A \rightarrow aS \mid \lambda$
$B \rightarrow bS \mid bB$
$C \rightarrow cC \mid \lambda$

We can see that the start symbol has a recursive derivation which can be removed by introducing a new start symbol S' and adding a new rule $S' \rightarrow S$. Hence, the new equivalent grammar is

$G' = (N, T, P', S')$ where
$N = \{S', S, A, B, C\}$
$T = \{a, b, c, \lambda\}$
$P' : S' \rightarrow S$
$S \rightarrow AB \mid AC$
$A \rightarrow aS \mid \lambda$
$B \rightarrow bS \mid bB$
$C \rightarrow cC \mid \lambda$

2 Eliminating lambda rules

A variable that can derive the null string is called **nullable**. A grammar wihtout nullable variables is called **noncontracting**. A grammar without nullable variables except $S \Rightarrow \lambda$ is called **essentially noncontracing**. We can eliminate the nullable variables by

1. identifying the nullable variables

2. replacing the rules that derive the null string. i.e. replacing

   $A \rightarrow \lambda$ and $B \rightarrow uAv$

   with

   $B \rightarrow uAv$ and $B \rightarrow uv$

2.1 Algorithm for identifying nullable variables

input: context-free grammar $G = (N, T, P, S)$
NULL := $\{A \mid A \rightarrow \lambda \in P\}$
repeat
PREV := NULL
for each variable $A \in N$ do
if there is an A rule $A \rightarrow w$ and $w \in$ PREV then
NULL := NULL $\cup \{A\}$
until NULL = PREV

After applying eliminating the lambda rules, the grammar is now ***essentially noncontracting***.

3 Eliminating the chain rules

We can eliminate the chain rules by

1. identifying the chain rules

2. replacing the chain rules. i.e. replacing

   $A \to B$ and $B \to w$

   with

   $A \to w$ and $B \to w$

3.1 Algorithm for identifying chain rules

input: essentially noncontracing context-free grammar $G = (N, T, P, S)$

```
CHAIN(A) := { A }
PREV := ∅
repeat
    NEW := CHAIN(A) − PREV
    PREV := CHAIN(A)
    for each variable B ∈ NEW do
        for each rule B → C do
            CHAIN(A) := CHAIN(A) ∪ { C }
until CHAIN(A) = PREV
```

4 Eliminating the useless symbols

For a variable to be **useful**, two conditions must be satisfied:

1. The variable must occur in a sential form of the grammar; that is it must occur in a string derivable from S.

2. There must be a derivation of a terminal string from the variable.

4.1 Algorithm for identify variables that derive terminal strings

input: context-free grammar $G = (N, T, P, S)$

```
TERM := { A | there is a rule A → w ∈ P with w ∈ T* }
repeat
    PREV := TERM
    for each variable A ∈ N do
        if there is an A rule A → w and w ∈ (PREV ∪ T)* then
            TERM := TERM ∪ { A }
until PREV = TERM
```

We then remove all the rules containing variables not in the set TERM.

4.2 Algorithm for identify reachable variables

input: context-free grammar $G = (N, T, P, S)$

REACH := S
PREV := $\emptyset$
repeat
    NEW := REACH $-$ PREV
    PREV := REACH
    for each variable $A \in$ NEW do
        for each rule $A \to w$ do
            add all variables in w to REACH
until REACH = PREV

By eliminating all variables not in REACH, we obtain a grammar that contains no recursive start symbol, no lambda rule except $S \to \lambda$, and no useless variables.

5 Chomsky Normal Form

A context-free grammar $G = (N, T, P, S)$ is in **Chomsky normal form** if each rule has one of the following forms:

1. $A \to BC$
2. $A \to a$
3. $S \to \lambda$

where $B, C \in N - \{S\}$.

The derivation tree associated with a derivation in a Chomsky normal form grammar is a binary tree.

Suppose that G has the following preperties:

i. The start symbol of G is non-recursive.

ii. G does not contain lambda rules other than $S \to \lambda$.

iii. G does not contain chain rules.

iv. G does not contain useless symbols

The rules satisfying these conditions has one of the following forms:

1. $S \to \lambda$
2. $A \to a$
3. $A \to w$ where $w \in ((T \cup N) - \{S\})^*$ and $|w| > 1$

We need only to change the rules with the last form:

1. Remove the terminal from the right-hand side.

2. Break the rules with length greater than 1 into a sequence of rules, each of whose right-hand side consists of two variables.

6 Removal of direct left recursion

We can remove the direct left recursion by replacing
$A \to Ax$ and $A \to y$
with
$A \to yB$, $B \to xB$, and $B \to \lambda$

7 Greibach Normal Form

A context-free grammar $G = (N, T, P, S)$ is in **Greibach normal form** if each rule has one of the following forms:

1. $A \to aA_1A_2...A_n$

2. $A \to a$

3. $S \to \lambda$

where $a \in T$ and $A_i \in N - \{S\}$ for $i = 1, 2, ..., n$.

The conversion of a Chomsky normal form grammar to Greibach normal form uses two rule transformation techniques:

References

[1] T. Pittman and J. Peters, *The Art of Compiler Design.* Prentice-Hall, 1992.

[2] T. A. Sudkamp, *Languages and Machines.* Addison-Wesley, 1997.