Project Three - Report

Converting Context-Free Grammar into Chomsky Normal Form

Nov.17 2019, Harbin Institute of Technology, Shenzhen

> Members:
>
> - **Champagne Kwan** *conversion algorithm designation, coding*
> - **Yanglin Liu** *test & report*

> Programming Language: **Python3**

Part1. Introduction

In this experiment, we try to implementing an algorithm to convert a given context-free grammar to corresponding Chomsky normal form.

In formal language theory, a context-free grammar G is said to be in Chomsky normal form if all of its production rules are of the form:

$$A \to BC$$
$$A \to a$$
$$S \to \epsilon$$

where A, B, and C are nonterminal symbols, a is a terminal symbol (a symbol that represents a constant value), S is the start symbol, and ε denotes the empty string.

We can know that all of production rules must be:

- only one nonterminal symbol at left part, and
- all candidate forms (right part of production rules) contain exactly two nonterminal symbol, or only one terminal symbol
- start symbol could product an empty string

Part2. Designation

2.1 brief description of conversion method

Here we referenced an example from textbook[1]. The original CFG is shown below. And we use colored text to represent updated parts: red for adding, cyan for deleting.

$$
\begin{aligned}
Original: S &\rightarrow ASA|aB \\
A &\rightarrow B|S \\
B &\rightarrow b|\epsilon
\end{aligned}
$$

> 1. **Start**: introduce a new start symbol to grammar:
>
> $$
> \begin{aligned}
> &Step1: introduce\ new\ start\ symbol \\
> &\quad {\color{red}S_0} \rightarrow S \\
> &\quad S \rightarrow ASA|aB \\
> &\quad A \rightarrow B|S \\
> &\quad B \rightarrow b|\epsilon
> \end{aligned}
> $$
>
> 2. **Epsilon Eliminating**: for each rule, eliminating any empty string in right part, and replace left symbol in other rules with non-empty right parts.

$$Step2 : eliminate\ B \rightarrow \epsilon,\ \ and\ add\ to\ other\ rules\ contains\ B$$

$$
\begin{aligned}
&S_0 \rightarrow S \\
&S \rightarrow ASA|aB|a \\
&A \rightarrow B|S|\epsilon \\
&B \rightarrow b|\epsilon
\end{aligned}
$$

$$Step3 : eliminate\ A \rightarrow \epsilon,\ \ and\ add\ to\ other\ rules\ contains\ A$$

$$
\begin{aligned}
&S_0 \rightarrow S \\
&S \rightarrow ASA|aB|a|SA|AS|S \\
&A \rightarrow B|S|\epsilon \\
&B \rightarrow b
\end{aligned}
$$

3. **Unit Eliminating**: remove unit rules (only one nonterminal in right parts), and replace left symbol in other rules with corresponding right parts.

> $$
> \begin{aligned}
> &Step4: remove\ S \to S\\
> &\quad S_0 \to S\\
> &\quad S \to ASA|aB|a|SA|AS|S\\
> &\quad A \to B|S\\
> &\quad B \to b
> \end{aligned}
> $$
>
> $$
> \begin{aligned}
> &Step5: eliminate\ S_0 \to S\ and\ replace\ S\ use\ corresponding\ right\ part\\
> &\quad S_0 \to S|ASA|aB|a|SA|AS\\
> &\quad S \to ASA|aB|a|SA|AS\\
> &\quad A \to B|S\\
> &\quad B \to b
> \end{aligned}
> $$
>
> $$
> \begin{aligned}
> &Step6: eliminate\ A \to B\ and\ replace\ B\ use\ corresponding\ right\ part\\
> &\quad S_0 \to ASA|aB|a|SA|AS\\
> &\quad S \to ASA|aB|a|SA|AS\\
> &\quad A \to B|S|b\\
> &\quad B \to b
> \end{aligned}
> $$
>
> $$
> \begin{aligned}
> &Step7: eliminate\ A \to S\ and\ replace\ S\ use\ corresponding\ right\ part\\
> &\quad S_0 \to ASA|aB|a|SA|AS\\
> &\quad S \to ASA|aB|a|SA|AS\\
> &\quad A \to S|b|ASA|aB|a|SA|AS\\
> &\quad B \to b
> \end{aligned}
> $$
>
> 4. **Folding Replacement of variables**: convert remaining rules which right parts have more than 2 nonterminals into which have exact 2 nonterminals. Here we use *folding strategy* shown below:
>
> $$
> \begin{aligned}
> &Folding\ Strategy\ example:\\
> &\quad A \to BCDE\ will\ be\ converted\ into\ 3\ rules\\
> \\
> &\qquad F \to BC\\
> &\qquad G \to FD\\
> &\qquad A \to GE
> \end{aligned}
> $$

In step 4, algorithm should ensure that left part should be same once right parts are same.

2.2 Designation of Entity Classes

Here we need three entity classes: *variable, production rule and grammar*. In source code, the data structure is:

> - class **Var**: *represents a variable, terminals or nonterminals*
>   - **symbol**: string type, used as identifier to represent.
>   - **is_nonterm**: boolean type, indicates if variable is a nonterminal symbol.
> - class **Product**: *represents a production rule*
>   - **left**: a Var object, represents left part of rule, we only need one variable, it's guaranteed by feature of context-free.
>   - **right**: list, represents all right parts of rule, elements of this list is list since a rule may contains many right parts and any part may contains many variables.
> - class **CFG**: *represents a context-free grammar*
>   - **start**: a Var object, represents start symbol of grammar.
>   - **nonterm**: set, contains all nonterminal symbols appeared in grammar.
>   - **term**: set, contains all terminal symbols appeared in grammar.
>   - **products**: list, all Product objects.
>   - **nonterminal_cand**: a list of constants, used in step 4 for folding replacement of variables mentioned in Part 2.1

2.3 Core Algorithm

Algorithm of Epsilon Eliminating is shown below, creating this algorithm is challenging since a nonterminal which derives empty string may appear many times in a given candidate form, for example: if two rules below is known

$$A \rightarrow BSB \quad and \quad B \rightarrow \epsilon$$

then three new candidate forms will be added into first rule because any B can be assgined an empty value.

$$A \to BSB|BS|SB|S$$

Thus the algorithm is designed to be a recursive procedure to handle all situations.

```
function EpsilonEliminating:
  while not reached end of candidate and current symbol is not given symbol:
    StackPush(symbol stack,current symbol)
    current symbol = next symbol
   if reached end of candidate:
    add all symbols in symbol stack into new candidates
    return
   StackPush(symbol stack,given symbol)
   # First recursion, given symbol is not assigned to be empty
   EpsilonEliminating() with next position
   while not StackEmpty(symbol stack) and StackTop(symbol stack) != given symbol:
    StackPop(symbol stack)
   StackTop(symbol stack)
   # Second recursion, given symbol is assigned to be empty
   EpsilonEliminating() with next position
```

Part3. Test

Here are two test cases used in testing, and results.

$$TestCase1: \ (from\ textbook)$$

$$S \to ASA|aB$$
$$A \to B|S$$
$$B \to b|\epsilon$$

$$TestCase2: \ (from\ website)$$

$$S \to ASB$$
$$A \to aAS|a|\epsilon$$
$$B \to SbS|A|bb$$

```
TestCase 1:
Non-Terminals:A,B,S
Terminals:a,b
Start:S
Products:
  S0→DA|CB|a|AS|SA
  S→DA|CB|a|AS|SA
  A→b|DA|CB|a|AS|SA
  B→b
  C→a
  D→AS


TestCase 2:
Non-Terminals:A,B,S
Terminals:a,b
Start:S
Products:
  S0→EB|SB|AS
  S→EB|SB|AS
  A→FS|a|CS
  B→GS|DD|FS|a|CS
  C→a
  D→b
  E→AS
  F→CA
  G→SD
```