Alternative Vidhi to Conversion of Cyclic CNF->GNF

Avinash Bansal
Assistant Professor (CSE)

GNIT, Mullana.
Ambala (Haryana), India

Abstract---In automata theory Greibach Normal Form shows that A->aV_n^*, where 'a' is terminal symbol and V_n is nonterminal symbol where * shows zero or more rates of V_n [1]. Most popular questions, conversion of following cyclic CNF into GNF are:

Question 1 S->AA | a, A->SS | b
Question 2 S->AB, A->BS | b, B->SA | a
Question 3 S->AB, A->BS | b, B->AS | a [1]

To solve these questions, we need two technical lemmas and required one or more another variable like Z_1. In these questions, we have cyclic nature of production called cyclic CNF. We have modified the same rule by which we get the more reliable answer with less number of productions in right hand side without using lemmas and any another variable. This above method can be applied on all problems by which we produce the GNF.

Keywords: GNF, Cyclic CNF, Automata, Normal Form, Grammar, Conversion CNF->GNF.

I. Preliminary

Each context free grammar can be converted in to Greibach Normal Form, that shows that A->aα, where 'a' € ∑(terminal symbol) and α € Vn* (nonterminal symbol). This conversion can be used to prove that every context-free language can be accepted by a non-deterministic pushdown automaton [1, 2]. We can understand the concept with the help of example.

Let us take Question 1.
S->AA | a, A->SS | b
Here's the grammar:
S->AA | a
A->SS | b
First rename the variables: put A1 for S and A2 for A, Now
A1->A2A2 | a
A2->A1A1 | b
After apply A2 in A1 we can see A1->bA2 | a, and A2->b are in required form but A1->A1A1A2 are not.
Apply Lemma 1: [1 pp 206]
A2->A2A2A1 | aA1 | b
So, now we have still one problem with production:
A2->A2A2A1
Apply Lemma 2: [1 pp 206]
As per lemma we add a new variable named as B. Then
A2->b | aA1 | bB | aA1B
B->A2A1 | A2A1B
So now our grammar looks like:
A1->A2A2 | a
A2->b | aA1 | bB | aA1B
B->A2A1 | A2A1B
Now we must fix A1, so that is only starts with terminals:
A1->bA2 | aA1A2 | bBA2 | aA1BA2 | a
Then we must B in a similar fashion (replacing initial occurrences of A2)
B->bA1 | aA1A1 | bBA1 | aA1BA1 | bA1B | aA1A1B | bBA1B | aA1BA1B and now we have the following grammar:
A1->bA2 | aA1A2 | bBA2 | aA1BA2 | a
A2->b | aA1 | bB | aA1B
B->bA1 | aA1A1 | bBA1 | aA1BA1 | bA1B | aA1A1B| bBA1B | aA1BA1B
This is in the required GNF [1].

II. Cyclic CNF Concept

We can understand the cylic CNF concept with the help of following example.

Example 1 S->AA | a, A->SS | b
Example 2 S->AB, A->BS | b, B->SA | a
Example 3 S->AB, A->BS | b, B->AS | a

In these example, we have cyclic nature of production e.g. if we take example 1, then we see; we have S->A and A->S first element of the both production create a cyclic form S->A->S. Similarly in example 2, we have the same property S->A, A->B and B->S creates a cycle S->A->B->S. In example 3, A->B and B ->A forms a cycle A->B->A

III. Vidhi to Conversion of Cyclic CNF->GNF

1. To Check the CNF grammar has cycle in it, directly or indirectly (e.g. question 1 have cycle)
2. If it so; then write it first (In question 1. S->A->S).
3. Take the cycle in reverse direction (e.g. in question 1 first A and then S ignore last S. Production is A->S).
4. Apply that value which is already in GNF form to step 3 order productions; till cycle start symbol. As in question 1 S->bA | a (because A->b is already in GNF form)
5. Now put that value of production back in the remaining reverse order direction like A->S in question 1. New production for A->bAS | aS | b (because reverse order direction is S->A)
6. Again put that value as per the order of step 3 (No change in that production which is already in GNF form. Order for this step is A->S). S->bA | a is already in GNF form. Now S->bASA | aSA | bA | a
7. Now production are in required GNF form, if any left then applies those productions in rest. We get the desired GNF grammar. Then Collect all production.
Like in Question 1
S->bASA | aSA | bA | a
A->bAS | aS | b

We can understand these rules with the help of Fig 1.

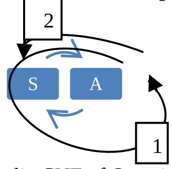

Fig. 1 Cyclic CNF of Question 1

As shown in Fig. 1, there are two loops having named 1 and 2 on it. Arrow marks on loop tell the direction as well as the ending point of the loop. Loop 1 complete five steps of algorithm. As per loop1 firstly put A's value in S and again that value put back in A. Now loop 2 complete 6th step of algorithm, in which A's value again put back in S. Finally collect all the production which is in required GNF form.

For more understanding these rules , we take question 2 and question 3. First we take question 2:
S->AB,   A->BS|b,   B->SA|a

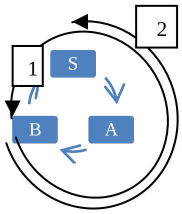

Fig. 2 Cyclic CNF of Question 2

As shown in Fig. 2 there is a cycle S->A->B->S where S is the starting and B is the last symbol. Then Loop 1 always starts from 'last symbol' and end with 'last symbol'. Similarly loop 2 always start with 'end symbol' and end with 'start symbol'. Algorithm steps are:

Step 1 Yes, cycle exist
Step 2 S->A->B->S (cycle). B will be counted in two steps, Step 3 and Step 5.
Step 3 B->A->S
Step 4 B->SA | a,   A->aS | b,
S->aSB | bB
Step 5 Remaining reverse order S->B then
B->aSBA | bBA | a
Step 6 Step 3 order (B->A->S)
A->aSBAS | bBAS | aS | b,
S->aSBASB | bBASB | aSB | bB
Step 7 Collect all Productions
S->aSBASB | bBASB | aSB | bB,
A->aSBAS | bBAS | aS | b,
B->aSBA | bBA | a

This is the required GNF

Now we take Question 3.
S->AB,   A->BS | b,   B->AS | a

Loop 1 of Fig. 3 complete five steps of algorithm and loop 3 execute 6th step of algorithm. After getting A's production in required GNF form, applies that value in S's production. Key point of algorithm is "puts that value which is already in GNF form", left the remaining for a moment. Residual production will be covered in the end of loop 1 that makes them in GNF.
Algorithm Steps are:

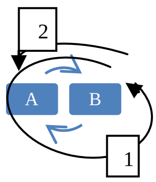

Fig. 3 Cyclic CNF of Question 3

Step 1 Yes, cycle exist.
Step 2 A->B->A (cycle). B will be counted in two steps, Step 3 and Step 5.
Step 3 B->A
Step 4 A->aS | b
Step 5 Remaining reverse order A->B
then B ->aSS | bS | a
Step 6 Step 3 order (B->A)
A->aSSS | bSS | aS | b
Step 7 Now the required answer as per algorithm.
S->aSSSB | bSSB | aSB | bB,
A->aSSS | bSS | aS | b,
B->aSS | bS | a

This is the required GNF. In the same way all cyclic CNF can be converted in to GNF form.

IV. Prove

The property of conversion grammar in automata theory says that, before and after conversion they produce exactly the same set of strings. We can check by designing any length of strings by both grammars. Both give the exactly same set.

V. Performance

As per the following Table 1 we can see, by this method we get less number of production on the right hand side.

TABLE I
GNF Production Comparison with Alternative Vidhi

S. No.	Question No.	No. of production by given method	No. of production by Alternative Vidhi
1	1	17	7
2	2	20	11
3	3	20	11

VI. Conclusions

Alternative Vidhi/way to conversion of cyclic CNF->GNF is more easy to understand and more easy to design. With the help of this 'conversion vidhi', we get very less production in the right hand side, so it is more reliable vidhi. With the help of this vidhi we get approximately deterministic way to produce the production/strings. This way is not only for cycle CNF to GNF, this can be apply on any context- free grammar.

Acknowledgment

I am grateful to the referees for some correction and their suggestions regarding the presentation of the result.

References

[1] K.L.P. Mishra, N. Chandrasekaran, *Theory of Computation*, PHI Third
Edition, August, 2008.
[2] Greibach Normal Form Available:
http://en.wikipedia.org/wiki/Greibach_normal_form