Automata and Grammars (BIE-AAG)

7. Context-free Grammars

Jan Holub

Department of Theoretical Computer Science
Faculty of Information Technology
Czech Technical University in Prague

Context-free Grammars

- Context-free grammars
  - They describe most of the syntactic structures of programming languages.
  - Algorithms for effective analysis of sentences of context-free languages are known.
- Parsing (Syntactic analysis):
  - Is the given string w generated by grammar G?
  - What is the structure of the string?

Ambiguous and Unambig. Grammars

Definition
Context-free grammar $G = (N, \Sigma, P, S)$ is *ambiguous*, if there is a sentence $w \in L(G)$ that is a result of at least two different parse trees. Otherwise the grammar is *unambiguous*.

Ambiguous and Unambig. Grammars

Example

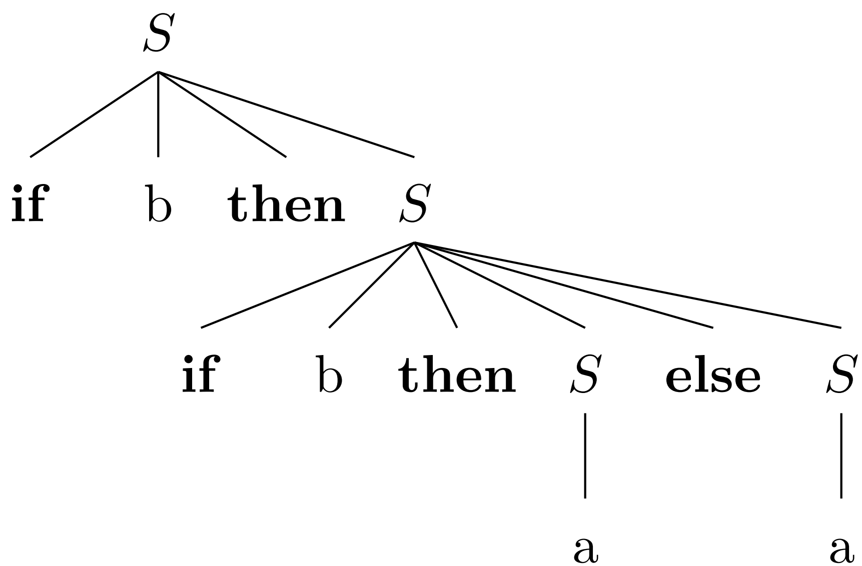
$G = (\{S\}, \{\mathrm{a}, \mathrm{b}, \mathbf{if}, \mathbf{then}, \mathbf{else}\}, P, S)$, where P:

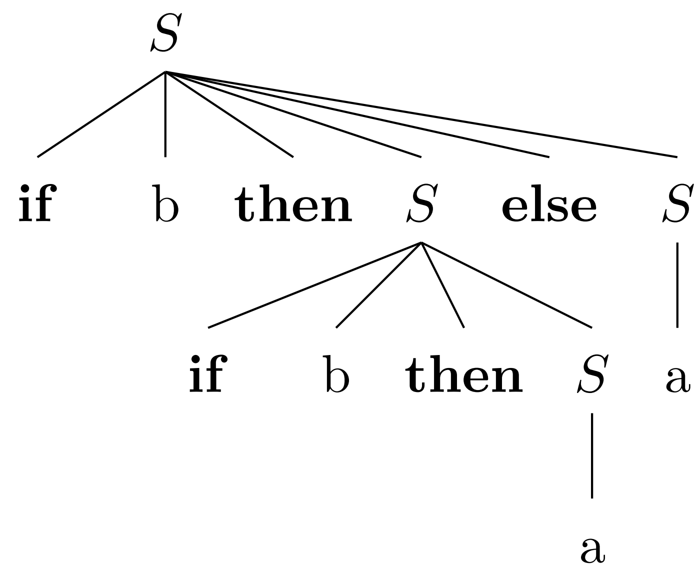
$S \rightarrow$ $\mathbf{if}$ b $\mathbf{then}$ S $\mathbf{else}$ S

$S \rightarrow$ $\mathbf{if}$ b $\mathbf{then}$ S

$S \rightarrow$ a

The grammar is ambiguous: $\mathbf{if}$ b $\mathbf{then}$ $\mathbf{if}$ b $\mathbf{then}$ a $\mathbf{else}$ a





Ambiguous and Unambig. Grammars

Example

$G = (\{A\}, \{a, b\}, P, A)$, where P:

$$A \to AA \qquad A \to b$$

CFG contains a rule $A \to AA$, therefore it is ambiguous. For a sentential form AAA there are two different parse trees.

Ambiguousness can be removed by replacing $A \to AA$ with rule

$A \to Ab$.

Ambiguous and Unambig. Grammars

- In many cases it is possible to remove the ambiguousness from the grammar.
- Inherently ambiguous languages:
  Cannot be generated by an unambiguous grammar.
- It is impossible to create an algorithm deciding whether the given CFG is ambiguous (by reduction from Post's correspondence problem).

Example

For the language in example in slide no. 4 the following unambiguous grammar can be constructed:

$G = (\{S_1, S_2\}, \{a, b, \mathbf{if}, \mathbf{then}, \mathbf{else}\}, P, S_1)$, where P:

$S_1 \to \mathbf{if}\ b\ \mathbf{then}\ S_1 \mid \mathbf{if}\ b\ \mathbf{then}\ S_2\ \mathbf{else}\ S_1 \mid a$

$S_2 \to \mathbf{if}\ b\ \mathbf{then}\ S_2\ \mathbf{else}\ S_2 \mid a$

Symbol $\mathbf{else}$ always belongs to the closest $\mathbf{then}$.

Transformations of CFG

Theorem
There is an algorithm that decides whether a language generated by the given context-free grammar is empty.

Algorithm Is $L(G)$ non-empty?
Input: Context-free grammar $G = (N, \Sigma, P, S)$.
Output: Yes, if $L(G) \neq \emptyset$, no, otherwise.

$N_0 \leftarrow \emptyset$; $i \leftarrow 0$
repeat
    $i \leftarrow i + 1;$
    $N_i \leftarrow \{A : A \in N, (A \to \alpha) \in P, \alpha \in (N_{i-1} \cup \Sigma)^*\} \cup N_{i-1}$
until $N_i = N_{i-1}$
$N_t \leftarrow N_i$
if $S \in N_t$ **then**
    return "Yes"
else
    return "No"
end if

Transformations of CFG

Example
$G = (\{S, A, B\}, \{a, b\}, \{S \to a, S \to A, A \to AB, B \to b\}, S)$.

$N_0 = \emptyset$
$N_1 = \{S, B\}$
$N_2 = \{S, B\}$
$N_1 = N_2 = N_t$
$S \in N_t \Rightarrow$ Grammar G generates a non-empty language.

Transformations of CFG

Definition
Symbol $X \in N \cup \Sigma$ is *unreachable* in context-free grammar $G = (N, \Sigma, P, S)$, if X does not appear in any sentential form, i.e. there is no derivation of the form $S \Rightarrow^* \alpha X \beta,\ \alpha, \beta \in (N \cup \Sigma)^*$.

Transformations of CFG

Algorithm Exclusion of unreachable symbols

Input: Context-free grammar $G = (N, \Sigma, P, S)$.
Output: CFG $G' = (N', \Sigma', P', S)$ such that $L(G') = L(G)$, $\forall X \in N' \cup \Sigma'$, $\exists \alpha, \beta \in (N' \cup \Sigma')^*$, $S \Rightarrow^* \alpha X \beta$.

$V_0 \leftarrow \{S\}$; $i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  $V_i \leftarrow \{X : X \in N \cup \Sigma, (A \rightarrow \alpha X \beta) \in P, A \in V_{i-1}\} \cup V_{i-1}$
until $V_i = V_{i-1}$
$N' \leftarrow V_i \cap N$
$\Sigma' \leftarrow V_i \cap \Sigma$
$P' \leftarrow \{A \rightarrow \alpha : A \in N', \alpha \in V_i{}^*, (A \rightarrow \alpha) \in P\}$
$G' \leftarrow (N', \Sigma', P', S)$
return G'

Transformations of CFG

Example

$G = (\{S, A, B\}, \{a, b\}, \{S \to a, S \to A, A \to AB, B \to b\}, S)$

$V_0 = \{S\}$
$V_1 = \{S\} \cup \{a, A\}$
$V_2 = \{S, A, a\} \cup \{B\}$
$V_3 = \{S, A, B, a\} \cup \{b\}$
$V_4 = \{S, A, B, a, b\}$

Transformations of CFG

Definition
Symbol $X \in N \cup \Sigma$ is *redundant* in $G = (N, \Sigma, P, S)$, if there does not exist $S \Rightarrow^* wXy \Rightarrow^* wxy$, where $w, x, y \in \Sigma^*$.

Transformations of CFG

Algorithm Exclusion of redundant symbols.

Input: Context-free grammar $G = (N, \Sigma, P, S)$, $L(G) \neq \emptyset$.
Output: CFG $G' = (N', \Sigma', P', S)$, $L(G') = L(G)$, $\forall X \in N' \cup \Sigma'$, $\exists \alpha, \beta, \gamma \in \Sigma'^* : S \Rightarrow^* \alpha X \beta$.

1: Using algorithm "Is $L(G)$ non-empty?" we get N_t. $G_1 \leftarrow (N_t, \Sigma, P_1, S)$, $P_1 \leftarrow \{A \to \alpha : A \in N_t, \alpha \in (N_t \cup \Sigma)^*, (A \to \alpha) \in P\}$.
2: Using algorithm "Exclusion of unreachable symbols" we exclude all unreachable symbols. We get $G' = (N', \Sigma', P', S)$.
3: **return** G'

Definition

Context-free grammar $G = (N, \Sigma, P, S)$ is *reduced*, if it contains no redundant symbols.

Transformations of CFG

Example

$G = (\{S, A, B\}, \{a, b\}, \{S \to a, S \to A, A \to AB, B \to b\}, S)$

Step 1:

$N_t = \{S, B\}$

$G_1 = (\{S, B\}, \{a, b\}, \{S \to a, B \to b\}, S)$

Step 2:

$V_0 = \{S\}$

$V_1 = \{S, a\}$

$V_2 = \{S, a\}$

$G' = (\{S\}, \{a\}, \{S \to a\}, S)$

Transformations of CFG

Theorem (Rule exclusion theorem, substitution theorem)
Let $G = (N, \Sigma, P, S)$ be a CFG and $(A \to \alpha B \beta) \in P, B \in N, A \neq B$, $\alpha, \beta \in (N \cup \Sigma)^*$.
Let $B \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_k$ be all rules in P with symbol B on the left-hand side.
Let $G' = (N, \Sigma, P', S)$, where
$P' = P \cup \{A \to \alpha\gamma_1\beta \mid \alpha\gamma_2\beta \mid \ldots \mid \alpha\gamma_k\beta\} \setminus \{A \to \alpha B \beta\}$.
Then $L(G) = L(G')$.

Transformations of CFG

Definition

Context-free grammar $G = (N, \Sigma, P, S)$ is *cycle-free*, if no derivation $A \Rightarrow^{+} A$ is possible for any $A \in N$.

Definition

Context-free grammar $G = (N, \Sigma, P, S)$ is ε*-rule free*, if

1. P contains no ε-rule, or
2. P contains only one ε-rule of form $S \to \varepsilon$ and S does not appear on the right hand side of any rule in P.

Definition

Context-free grammar $G = (N, \Sigma, P, S)$ is *proper*, if it is cycle free, ε-rule free, and it contains no redundant symbols.

Transformations of CFG

Algorithm Exclusion of ε-rules.

Input: Context-free grammar $G = (N, \Sigma, P, S)$, $L(G) \neq \emptyset$.
Output: CFG without ε-rules, $L(G') = L(G)$.

$N_0 \leftarrow \emptyset$; $i \leftarrow 0$
repeat
    $i \leftarrow i + 1$
    $N_i \leftarrow \{A : (A \to \alpha) \in P, \alpha \in N_{i-1}^*\}$
until $N_{i-1} = N_i$
$N_\varepsilon \leftarrow N_i$
$P' \leftarrow \{A \to \alpha_1\alpha_2 : (A \to \alpha_1 X \alpha_2) \in P, X \in N_\varepsilon, \alpha_1, \alpha_2 \in (N \cup \Sigma)^*, \alpha_1\alpha_2 \neq \varepsilon\}$
$P' \leftarrow P' \cup (P \setminus \{A \to \varepsilon : (A \to \varepsilon) \in P, A \in N\})$
if $S \in N_\varepsilon$ **then**
    $P' \leftarrow P' \cup \{S' \to \varepsilon, S' \to S\}, S' \notin N$
    $N' \leftarrow N' \cup \{S'\}$
else
    $S' \leftarrow S$
end if
$G' \leftarrow (N', \Sigma', P', S')$
return G'

Transformations of CFG

Algorithm Exclusion of simple rules.

Input: Context-free grammar $G = (N, \Sigma, P, S)$, $L(G) \neq \emptyset$.

Output: CFG without simple rules, $L(G') = L(G)$.

for $A \in N$ **do**
    $N_0 \leftarrow \{A\}$; $i \leftarrow 0$
    repeat
        $i \leftarrow i + 1$
        $N_{i-1} \leftarrow \{C : (B \rightarrow C) \in P, B \in N_{i-1}\} \cup N_{i-1}$
    until $N_{i-1} = N_i$
    $N_A \leftarrow N_i$
end for
$P' \leftarrow \emptyset$
for $A \in N$ **do**
    $P' \leftarrow P' \cup \{A \rightarrow \alpha : (B \rightarrow \alpha) \in P, B \in N_A, \alpha \in ((N \cup \Sigma)^* \setminus N)\}$
end for
$G' \leftarrow (N, \Sigma', P', S)$
return G'

Transformations of CFG

Theorem
If context-free grammar $G = (N, \Sigma, P, S)$ has no ε-rules and simple rules, then it is cycle-free.

Theorem
If L is a context-free language, then it can be generated by some proper grammar G.

Chomsky normal form

Definition (Chomsky normal form)
CFG $G = (N, \Sigma, P, S)$ is in Chomsky normal form if every rule in P is in one of the following forms:

1. $A \to BC$, where $A, B, C \in N$.
2. $A \to a$, where $a \in \Sigma, A \in N$.
3. $S \to \varepsilon$, if $\varepsilon \in L(G)$ and S does not appear on the right-hand side of any of the rules.

Theorem
Let L be a context-free language. L is a language generated by a grammar in Chomsky normal form.

Chomsky normal form

Algorithm Conversion of CFG to Chomsky normal form

Input: Proper context-free grammar $G = (N, \Sigma, P, S)$ without simple rules.
Output: CFG G' in Chomsky normal form, $L(G) = L(G')$

$N' \leftarrow \emptyset$
$P' \leftarrow \{A \rightarrow BC : (A \rightarrow BC) \in P, A, B, C \in N\}$
$P' \leftarrow P' \cup \{A \rightarrow a : (A \rightarrow a) \in P, A \in N, a \in \Sigma\}$
$P' \leftarrow P' \cup \{S \rightarrow \varepsilon : (S \rightarrow \varepsilon) \in P\}$
for $(A \rightarrow X_1 X_2 \dots X_k) \in P, k > 2, X_i \in (N \cup \Sigma)$ **do**
    $N' \leftarrow N' \cup \{Y_{X_i \dots X_k} : 1 < i \leq k, Y_{X_i \dots X_k} \notin (N \cup N')\}$
    $P' \leftarrow P' \cup \{A \rightarrow X'_1 Y_{X_2 \dots X_k}, Y_{X_2 \dots X_k} \rightarrow X'_2 Y_{X_3 \dots X_k}, \cdots, Y_{X_{k-2} \dots X_k} \rightarrow X'_{k-2} Y_{X_{k-1} X_k}, Y_{X_{k-1} X_k} \rightarrow X'_{k-1} X'_k\}$,
    if $X_i \in N$, then $X'_i = X_i$, otherwise, $N' \leftarrow N' \cup \{X'_i : X_i \in \Sigma, 1 \leq i \leq k\}$;
    $P' \leftarrow P' \cup \{X'_i \rightarrow X_i : X_i \in \Sigma, 1 \leq i \leq k\}$
end for
for $(A \rightarrow X_1 X_2) \in P, X_1 \in \Sigma \vee X_2 \in \Sigma$ **do**
    $P' \leftarrow P' \cup \{A \rightarrow X'_1 X'_2\}$, if $X_i \in N, i \in \{1, 2\}$, then $X'_i = X_i$, otherwise, $N' \leftarrow N' \cup \{X'_i : X_i \in \Sigma, i \in \{1, 2\}\}$; $P' \leftarrow P' \cup \{X'_i \rightarrow X_i : X_i \in \Sigma, i \in \{1, 2\}\}$
end for
$G' \leftarrow (N' \cup N, \Sigma, P', S)$
return G'

Chomsky normal form

Example

Proper CFG $G = (\{S, A, B\}, \{a, b\}, P, S)$, where P:

$S \rightarrow aAB \mid BA$
$A \rightarrow BBB \mid a$
$B \rightarrow AS \mid b$

step 2:	$S \rightarrow BA$, $B \rightarrow AS$	inserted into P'.
step 3:	$A \rightarrow a$, $B \rightarrow b$	inserted into P'.
step 5–8:	$(S \rightarrow aAB) \in P \Rightarrow S \rightarrow a'Y_{AB}$, $Y_{AB} \rightarrow AB$	inserted into P'.
	$(A \rightarrow BBB) \in P \Rightarrow A \rightarrow BY_{BB}$, $Y_{BB} \rightarrow BB$	inserted into P'.
	$a' \rightarrow a$	inserted into P'.
step 12:	$P' = \{S \rightarrow a'Y_{AB} \mid BA$, $A \rightarrow BY_{BB} \mid a$, $B \rightarrow AS \mid b$, $Y_{AB} \rightarrow AB$, $Y_{BB} \rightarrow BB$, $a' \rightarrow a\}$	
	$N' = N \cup \{Y_{AB}, Y_{BB}, a'\}$	

Cocke-Younger-Kasami algorithm

Algorithm Cocke-Younger-Kasami (CYK)

Input: CFG $G = (N, \Sigma, P, S)$ in Chomsky normal form, $x = x_1.x_2 \ldots x_n \in \Sigma^*, x_i \in \Sigma$.

Output: Answer, whether $x \in L(G)$.

$P[i,j] \leftarrow \emptyset$, $\forall i, j \in \{1, 2, \ldots, n\}$ ▷ Initialize array
$P[i,1] \leftarrow P[i,1] \cup \{A\}$, $\forall i \in \{1, 2, \ldots, n\}$, $\forall A \in N$, $(A \rightarrow x_i) \in P$
for $i \in \{2, \ldots, n\}$ **do** ▷ Length of span
    for $j \in \{1, \ldots, n-i+1\}$ **do** ▷ Start of span
        for $k \in \{1, \ldots, i-1\}$ **do** ▷ Partition of span
            $P[j,i] \leftarrow P[j,i] \cup \{A\}$, $\forall \{A \rightarrow BC\} \in P$, $B \in P[j,k] \wedge C \in P[j+k, i-k]$
        end for
    end for
end for
if $S \in P[1,n]$ **then**
    return $x \in L(G)$
else
    return $x \notin L(G)$
end if

Cocke-Younger-Kasami algorithm

Example

$x = aaba$, $G = (\{S, A, B, C\}, \{a, b\}, P, S)$, where P:

$S \rightarrow AB \mid BC$

$A \rightarrow BA \mid a$

$B \rightarrow CC \mid b$

$C \rightarrow AB \mid a$

4				
3				
2				
1				
	a	a	b	a

Cocke-Younger-Kasami algorithm

Theorem
For each CFG $G = (N, \Sigma, P, S)$ in Chomsky normal form there exists an algorithm such that for given input string $x \in \Sigma^*$ of length n it decides in time $\mathcal{O}(n^3)$, whether $x \in L(G)$.

Recursive CFG

Definition
Nonterminal symbol A in CFG $G = (N, \Sigma, P, S)$ is called *recursive*, if there exists a derivation $A \Rightarrow^+ \alpha A \beta$ for some α and $\beta \in (N \cup \Sigma)^*$. If $\alpha = \varepsilon$, then A is called *left-recursive symbol*, similarly if $\beta = \varepsilon$, then A is called *right-recursive symbol*.
Grammar with at least one (right-)left-recursive nonterminal is called *(right-)left-recursive*.
Grammar in which at least one nonterminal symbol is recursive is called *recursive*.

Recursive CFG

Theorem (On excluding left recursion from single rules)
Let $G = (N, \Sigma, P, S)$ be a CFG, in which
$A \to A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$
are all rules in P with nonterminal symbol A on the left-hand side and no β_i begins with symbol A, $\forall i \in \{1, 2, \dots, n\}$.
Let $G' = (N \cup \{A'\}, \Sigma, P', S)$ be a CFG, where P' is the set P in which all above mentioned rules are replaced by these rules:
$A \to \beta_1 \mid \beta_2 \mid \dots \mid \beta_n \mid \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_n A'$
$A' \to \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_m \mid \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A'$,
where A' is a new nonterminal symbol, $A' \notin N$.
Then $L(G') = L(G)$.

Recursive CFG

Example

$G = (\{E, T, F\}, \{+, *, (,), a\}, P, E)$, where P:
$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid a$.

We remove left recursion:
$E \rightarrow T \mid TE'$
$E' \rightarrow +T \mid +TE'$
$T \rightarrow F \mid FT'$
$T' \rightarrow *F \mid *FT'$
$F \rightarrow (E) \mid a$

$G' = (\{E, E', T', T, F\}, \{+, *, (,), a\}, P', E)$, where P' is the set of rules given above.

Recursive CFG

Theorem
Every context-free language can be generated by a grammar that does not contain left recursion.

Recursive CFG

Algorithm Exclusion of left recursion.

Input: Proper CFG $G = (N, \Sigma, P, S)$ without simple rules.
Output: CFG G' without left recursion, $L(G) = L(G')$.

We choose ordering $N = \{A_1, \dots, A_r\}$
for $i \in \{1, \dots, r\}$ **do**
  for $j \in \{1, \dots, i-1\}$ **do**
    If $A_j \rightarrow \beta_1 \mid \dots \mid \beta_m$ are all the rules of P with $A_j \in N$ on the left-hand side, we replace all rules of form $A_i \rightarrow A_j\alpha$ by rules $A_i \rightarrow \beta_1\alpha \mid \dots \mid \beta_m\alpha$
  end for
  All rules $A_i \rightarrow A_i\alpha_1 \mid \dots \mid A_i\alpha_m \mid \beta_1 \mid \dots \mid \beta_n$ from P with nonterminal symbol A_i on the left-hand side, where no β_j begins with a nonterminal symbol A_k for $k \leq i$, are replaced by these rules:
  $A_i \rightarrow \beta_1 \mid \dots \mid \beta_n \mid \beta_1 A_i' \mid \dots \mid \beta_n A_i'$,
  $A_i' \rightarrow \alpha_1 \mid \dots \mid \alpha_m \mid \alpha_1 A_i' \mid \dots \mid \alpha_m A_i'$, where A_i' is a new nonterminal symbol.
end for

Recursive CFG

Example

$G = (\{A, B, C\}, \{a, b\}, P, A)$, where P:
$A \to BC \mid a$
$B \to CA \mid Ab$
$C \to AB \mid CC \mid a$.

We apply Algorithm "Exclusion of left recursion" to this grammar.
$A_1 \leftarrow A, A_2 \leftarrow B, A_3 \leftarrow C$.
Step 6: $(i = 1)$ without any changes.
Step 4: $(i = 2, j = 1)$
After substituting A we get these rules for B: $B \to CA \mid BCb \mid ab$.
Step 6: We remove left recursion at the symbol B:
$B \to CA \mid ab \mid CAB' \mid abB'$
$B' \to Cb \mid CbB'$
Step 4: $(i = 3, j = 1)$
$C \to BCB \mid aB \mid CC \mid a$

Recursive CFG

Example (continued)

Step 4: $(i = 3, j = 2)$

$C \to CACB \mid abCB \mid CAB'CB \mid abB'CB \mid aB \mid CC \mid a$

Step 6: $(i = 3)$

$C \to abCB \mid abB'CB \mid aB \mid a \mid abCBC' \mid abB'CBC' \mid aBC' \mid aC'$

$C' \to ACB \mid AB'CB \mid C \mid ACBC' \mid AB'CBC' \mid CC'$

The resulting grammar is $G' = (\{A, B, C, B', C'\}, \{a, b\}, P', A)$, where P':

$A \to BC \mid a$

$B \to CA \mid ab \mid CAB' \mid abB'$

$B' \to Cb \mid CbB'$

$C \to abCB \mid abB'CB \mid aB \mid a \mid abCBC' \mid abB'CBC' \mid aBC' \mid aC'$

$C' \to ACB \mid AB'CB \mid C \mid ACBC' \mid AB'CBC' \mid CC'$