MA/CSSE 474
Theory of Computation

More about Ambiguity Removal

Normal Forms (Chomsky and Greibach)

Pushdown Automata (PDA) Intro

PDA examples

Your Questions?

- Previous class days' material
- Reading Assignments
- HW10 or 11 problems
- Anything else

Continue with Ambiguity Removal

- Remove ε-rules (done last time)
- Eliminate symmetric rules to control precedence and association
- Deal with optional suffixes, such as **if … else …**

Recap: An Example

$G = \{\{S, T, A, B, C, \text{a}, \text{b}, \text{c}\}, \{\text{a}, \text{b}, \text{c}\}, R, S\}$,

$R = \{\ S \rightarrow \text{a}T\text{a}$
$T \rightarrow ABC$
$A \rightarrow \text{a}A \mid C$
$B \rightarrow B\text{b} \mid C$
$C \rightarrow \text{c} \mid \varepsilon\ \}$

Recall:
After this algorithm runs, L(G') = L(G) – {ε})

removeEps(*G*: cfg) =
1. Let $G' = G$.
2. Find the set N of nullable nonterminals in G'.
3. Repeat until G' contains no modifiable rules that haven't been processed:
   Given the rule $P \rightarrow \alpha Q\beta$, where $Q \in N$, add the rule $P \rightarrow \alpha\beta$ if it is not already present and if $\alpha\beta \neq \varepsilon$ and if $P \neq \alpha\beta$.
4. Delete from G' all rules of the form $X \rightarrow \varepsilon$.
5. Return G'.

What If $\varepsilon \in L$?

atmostoneEps(G: cfg) =

1. G'' = *removeEps*(G).
2. If S_G is nullable then /* i. e., $\varepsilon \in L(G)$
    - 2.1 Create in G'' a new start symbol S^*.
    - 2.2 Add to $R_{G''}$ the two rules:

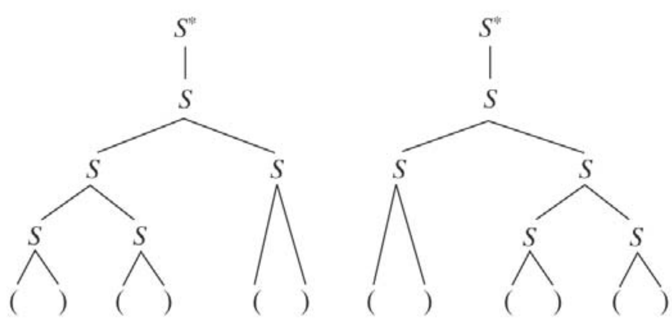
      $S^* \rightarrow \varepsilon$

      $S^* \rightarrow S_G$.
3. Return G''.

But There Can Still Be Ambiguity

$S^* \rightarrow \varepsilon$
$S^* \rightarrow S$
$S \rightarrow SS$
$S \rightarrow (S)$
$S \rightarrow ()$

What about ()()() ?



Eliminating Symmetric Recursive Rules

$S^* \rightarrow \varepsilon$
$S^* \rightarrow S$
$S \rightarrow SS$
$S \rightarrow (S)$
$S \rightarrow ()$

Replace $S \rightarrow SS$ with one of:

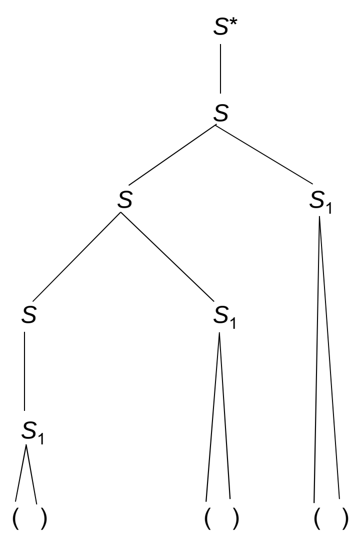
$S \rightarrow SS_1$ /* force branching to the left
$S \rightarrow S_1S$ /* force branching to the right

So we get:

$S^* \rightarrow \varepsilon$
$S^* \rightarrow S$
$S \rightarrow SS_1$
$S \rightarrow S_1$
$S_1 \rightarrow (S)$
$S_1 \rightarrow ()$

Eliminating Symmetric Recursive Rules

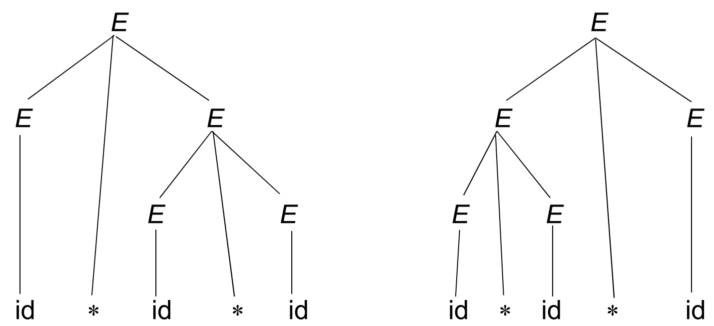
$S^* \rightarrow \varepsilon$
$S^* \rightarrow S$
$S \rightarrow SS_1$
$S \rightarrow S_1$
$S_1 \rightarrow (S)$
$S_1 \rightarrow ()$



Arithmetic Expressions

$E \to E + E$
$E \to E * E$
$E \to (E)$
$E \to \texttt{id}$

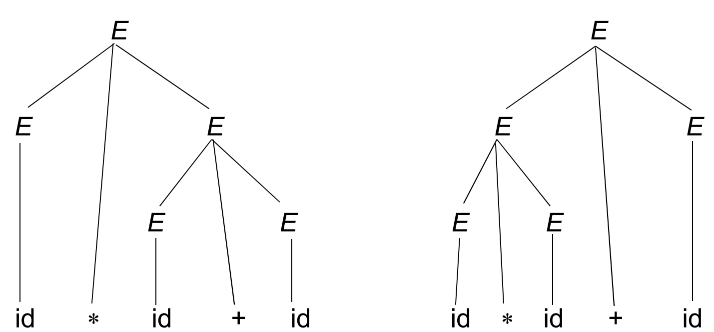
Problem 1: Associativity



Arithmetic Expressions

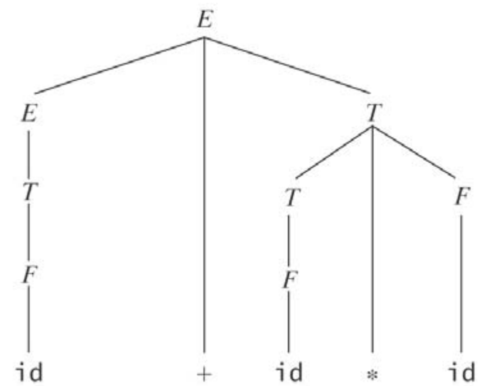
$E \to E + E$
$E \to E * E$
$E \to (E)$
$E \to \texttt{id}$

Problem 2: Precedence



Arithmetic Expressions - A Better Way

$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow \text{id}$



Ambiguous Attachment

The dangling else problem:

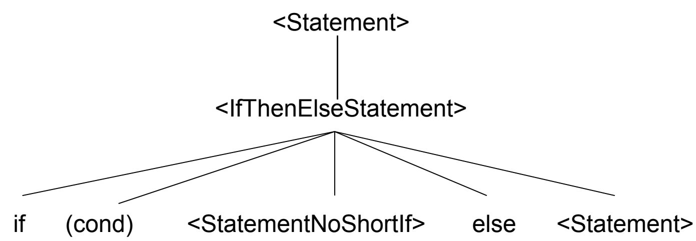
<stmt> ::= `if` <cond> `then` <stmt>
<stmt> ::= `if` <cond> `then` <stmt> `else` <stmt>

Consider:

`if` cond_1 `then` `if` cond_2 `then` st_1 `else` st_2

The Java Fix

<Statement> ::= <IfThenStatement> | <IfThenElseStatement> |
<IfThenElseStatementNoShortIf>
<StatementNoShortIf> ::= <block> |
<IfThenElseStatementNoShortIf> | …
<IfThenStatement> ::= `if` (<Expression>) <Statement>
<IfThenElseStatement> ::= `if` (<Expression>)
<StatementNoShortIf> `else` <Statement>
<IfThenElseStatementNoShortIf> ::=
`if` (<Expression>) <StatementNoShortIf>
`else` <StatementNoShortIf>



Going Too Far (removing Ambiguity)

S → *NP VP*
NP → the *Nominal* | *Nominal* | *ProperNoun* | *NP PP*
Nominal → *N* | *Adjs N*
N → `cat` | `girl` | `dogs` | `ball` | `chocolate` | `bat`
ProperNoun → `Chris` | `Fluffy`
Adjs → *Adj Adjs* | *Adj*
Adj → `young` | `older` | `smart`
VP → *V* | *V NP* | *VP PP*
V → `like` | `likes` | `thinks` | `hits`
PP → *Prep NP*
Prep → `with`

- `Chris likes the girl with the cat.`
- `Chris shot the bear with a rifle.`

Going Too Far

- `Chris likes the girl with the cat.`
- `Chris shot the bear with a rifle.`

- `Chris shot the bear with a rifle.`

Normal Forms

A ***normal form*** *F* for a set *C* of data objects is a form, i.e., a set of syntactically valid objects, with the following two properties:

- For every element *c* of *C*, except possibly a finite set of special cases, there exists some element *f* of *F* such that *f* is equivalent to *c* with respect to some set of tasks.
- *F* is simpler than the original form in which the elements of *C* are written. By “simpler” we mean that at least some tasks are easier to perform on elements of *F* than they would be on elements of *C*.

Normal Form Examples

- **Disjunctive normal form** for database queries so that they can be entered in a query-by-example grid.

- **Jordan normal form** for a square matrix, in which the matrix is almost diagonal in the sense that its only non-zero entries lie on the diagonal and the superdiagonal.

- **Various normal forms for grammars** to support specific parsing techniques.

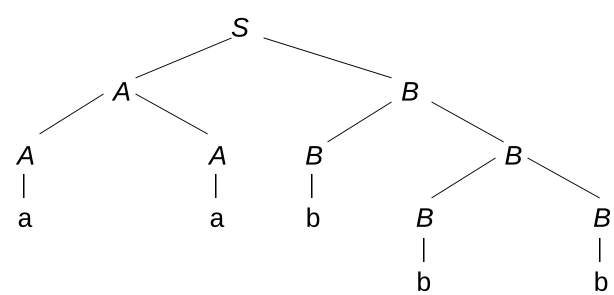
Normal Forms for Grammars

Chomsky Normal Form, in which all rules are of one of the following two forms:

- $X \rightarrow a$, where $a \in \Sigma$, or
- $X \rightarrow BC$, where B and C are elements of $V - \Sigma$.

Advantages:

- Parsers can use binary trees.
- Bounds on length of derivations (what are they?)



Normal Forms for Grammars

Greibach Normal Form, in which all rules are of the following form:

- $X \rightarrow a\, \beta$, where $a \in \Sigma$ and $\beta \in (V - \Sigma)^*$.

Advantages:

- Bounds on length of derivations (what are they?)
- Greibach normal form grammars can easily be converted to pushdown automata with no ε-transitions. This is useful because such PDAs are guaranteed to halt.

Theorems: Normal Forms Exist

Theorem: Given a CFG G, there exists an equivalent Chomsky normal form grammar G_C such that:

$$L(G_C) = L(G) - \{\varepsilon\}.$$

Proof: The proof is by construction.

Details of Chomsky conversion are complex but straightforward; I leave them for you to read in Chapter 11 and/or in the last 18 slides from today.

Theorem: Given a CFG G, there exists an equivalent Greibach normal form grammar G_G such that:

$$L(G_G) = L(G) - \{\varepsilon\}.$$

Proof: The proof is also by construction.

Details of Greibach conversion are more complex but still straightforward; I leave them for you to read in Appendix D if you wish (not req'd).

The Price of Normal Forms

$E \to E + E$
$E \to (E)$
$E \to \mathtt{id}$

Converting to Chomsky normal form:

$E \to E\ E'$
$E' \to P\ E$
$E \to L\ E''$
$E'' \to E\ R$
$E \to \mathtt{id}$
$L\ \to ($
$R\ \to)$
$P\ \to +$

Conversion doesn't change weak generative capacity but it may change strong generative capacity.

Pushdown Automata

Comparing Regular and Context-Free Languages

Regular Languages	Context-Free Languages
• regular exprs. or	
• regular grammars	• context-free grammars
• recognize	• parse (use a PDA)

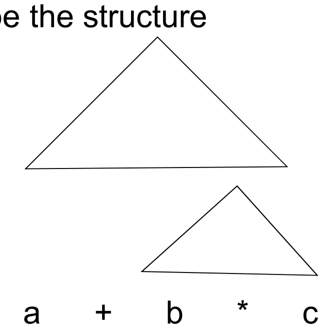
Recognizing Context-Free Languages

Two notions of recognition:

(1) Say yes or no, just like with FSMs

(2) Say yes or no, AND

if yes, describe the structure



Definition of a Pushdown Automaton

$M = (K, \Sigma, \Gamma, \Delta, s, A)$, where:

- K is a finite set of states
- Σ is the input alphabet
- Γ is the stack alphabet
- $s \in K$ is the initial state
- $A \subseteq K$ is the set of accepting states, and
- Δ is the transition relation. It is a finite subset of

Σ and Γ are not necessarily disjoint

$$\underbrace{(K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^*)}\ \times\ \underbrace{(K \times \Gamma^*)}$$

state	input symbol or ε	string of symbols to pop from top	state	string of symbols to push on stack

Definition of a Pushdown Automaton

A **configuration** of M is an element of $K \times \Sigma^* \times \Gamma^*$.

The **initial configuration** of M is (s, w, ε), where w is the input string.

Manipulating the Stack

c
a
b

will be written as cab

If $c_1c_2\ldots c_n$ is pushed onto the stack:

c_1
c_2
c_n
c
a
b

$c_1c_2\ldots c_n$cab

Yields

Let c be any element of $\Sigma \cup \{\varepsilon\}$,
Let γ_1, γ_2 and γ be any elements of Γ^*, and
Let w be any element of Σ^*.

Then:
$(q_1, cw, \gamma_1\gamma) \vdash_M (q_2, w, \gamma_2\gamma)$ iff $((q_1, c, \gamma_1), (q_2, \gamma_2)) \in \Delta$.

Let $\vdash_M^*$ be the reflexive, transitive closure of $\vdash_M$.

C_1 ***yields*** configuration C_2 iff $C_1 \vdash_M^* C_2$

Computations

A ***computation*** by M is a finite sequence of configurations C_0, C_1, …, C_n for some $n \geq 0$ such that:

- C_0 is an initial configuration,
- C_n is of the form (q, ε, γ), for some state $q \in K_M$ and some string γ in Γ^*, and
- $C_0 \vdash_M C_1 \vdash_M C_2 \vdash_M \ldots \vdash_M C_n$.

Nondeterminism

If M is in some configuration (q_1, s, γ) it is possible that:

- Δ contains exactly one transition that matches.
- Δ contains more than one transition that matches.
- Δ contains no transition that matches.

Accepting

A computation C of M is an ***accepting computation*** iff:

- $C = (s, w, \varepsilon) \vdash_M^* (q, \varepsilon, \varepsilon)$, and
- $q \in A$.

M ***accepts*** a string w iff at least one of its computations accepts.

Other paths may:

- Read all the input and halt in a nonaccepting state,
- Read all the input and halt in an accepting state with the stack not empty,
- Loop forever and never finish reading the input, or
- Reach a dead end where no more input can be read.

The ***language accepted by*** M, denoted $L(M)$, is the set of all strings accepted by M.

Rejecting

A computation C of M is a ***rejecting computation*** iff:

- $C = (s, w, \varepsilon) \mid\vdash_M^* (q, \varepsilon, \alpha)$,
- C is not an accepting computation, and
- M has no moves that it can make from (q, ε, α).

M ***rejects*** a string w iff all of its computations reject.

Note that it is possible that, on input w, M neither accepts nor rejects.

Details of CNF conversion

- The remainder of the slides give an overview.
- More details are in Chapter 11.
- We will not cover these details in class.

Converting to a Normal Form

1. Apply some transformation to *G* to get rid of undesirable property 1. Show that the language generated by *G* is unchanged.

2. Apply another transformation to *G* to get rid of undesirable property 2. Show that the language generated by *G* is unchanged *and* that undesirable property 1 has not been reintroduced.

3. Continue until the grammar is in the desired form.

Rule Substitution

$X \to \mathrm{a}Y\mathrm{c}$
$Y \to \mathrm{b}$
$Y \to ZZ$

We can replace the X rule with the rules:

$X \to \mathrm{abc}$
$X \to \mathrm{a}ZZ\mathrm{c}$

$$X \Rightarrow \mathrm{a}Y\mathrm{c} \Rightarrow \mathrm{a}ZZ\mathrm{c}$$

Rule Substitution

Theorem: Let G contain the rules:

$$X \to \alpha Y \beta \quad \text{and} \quad Y \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n ,$$

Replace $X \to \alpha Y \beta$ by:

$$X \to \alpha\gamma_1\beta, \quad X \to \alpha\gamma_2\beta, \quad \ldots, \quad X \to \alpha\gamma_n\beta.$$

The new grammar G' will be equivalent to G.

Details of Conversion to CNF

- The rest of these slides summarize the CNF conversion
- More detail is given in Chapter 11 of the textbook
- We will not discuss this conversion process in class.

Rule Substitution

Replace $X \rightarrow \alpha Y \beta$ by:

$X \rightarrow \alpha\gamma_1\beta, \quad X \rightarrow \alpha\gamma_2\beta, \quad \ldots, X \rightarrow \alpha\gamma_n\beta.$

Proof:

● Every string in $L(G)$ is also in $L(G')$:

If $X \rightarrow \alpha Y \beta$ is not used, then use same derivation.
If it is used, then one derivation is:
$S \Rightarrow \ldots \Rightarrow \delta X \phi \Rightarrow \delta\alpha Y \beta\phi \Rightarrow \delta\alpha\gamma_k\beta\phi \Rightarrow \ldots \Rightarrow w$

Use this one instead:
$S \Rightarrow \ldots \Rightarrow \delta X \phi \Rightarrow \quad \delta\alpha\gamma_k\beta\phi \Rightarrow \ldots \Rightarrow w$

● Every string in $L(G')$ is also in $L(G)$: Every new rule can be simulated by old rules.

Convert to Chomsky Normal Form

1. Remove all ε-rules, using the algorithm *removeEps*.
2. Remove all unit productions (rules of the form $A \rightarrow B$).
3. Remove all rules whose right hand sides have length greater than 1 and include a terminal:

   (e.g., $A \rightarrow \mathrm{a}B$ or $A \rightarrow B\mathrm{a}C$)
4. Remove all rules whose right hand sides have length greater than 2:

   (e.g., $A \rightarrow BCDE$)

Recap: Removing ε-Productions

Remove all ε productions:

(1) If there is a rule $P \rightarrow \alpha Q \beta$ and Q is nullable,

Then: Add the rule $P \rightarrow \alpha\beta$.

(2) Delete all rules $Q \rightarrow \varepsilon$.

Removing ε-Productions

Example:

$S \rightarrow aA$
$A \rightarrow B \mid CDC$
$B \rightarrow \varepsilon$
$B \rightarrow a$
$C \rightarrow BD$
$D \rightarrow b$
$D \rightarrow \varepsilon$

Unit Productions

A ***unit production*** is a rule whose right-hand side consists of a single nonterminal symbol.

Example:

$S \rightarrow X\ Y$
$X \rightarrow A$
$A \rightarrow B \mid a$
$B \rightarrow b$
$Y \rightarrow T$
$T \rightarrow Y \mid c$

Removing Unit Productions

removeUnits(*G*) =

1. Let G' = G.
2. Until no unit productions remain in G' do:
   - 2.1 Choose some unit production $X \rightarrow Y$.
   - 2.2 Remove it from G'.
   - 2.3 Consider only rules that still remain. For every rule $Y \rightarrow \beta$, where $\beta \in V^*$, do:
     - Add to G' the rule $X \rightarrow \beta$ unless it is a rule that has already been removed once.
3. Return G'.

After removing epsilon productions and unit productions, all rules whose right hand sides have length 1 are in Chomsky Normal Form.

Removing Unit Productions

removeUnits(*G*) =

1. Let G' = G.
2. Until no unit productions remain in G' do:
   - 2.1 Choose some unit production $X \rightarrow Y$.
   - 2.2 Remove it from G'.
   - 2.3 Consider only rules that still remain. For every rule $Y \rightarrow \beta$, where $\beta \in V^*$, do:
     - Add to G' the rule $X \rightarrow \beta$ unless it is a rule that has already been removed once.
3. Return G'.

Example:

$S \rightarrow X\ Y$
$X \rightarrow A$
$A \rightarrow B \mid \mathrm{a}$
$B \rightarrow \mathrm{b}$
$Y \rightarrow T$
$T \rightarrow Y \mid \mathrm{c}$

Mixed Rules

removeMixed(G) =

1. Let $G' = G$.
2. Create a new nonterminal T_a for each terminal a in Σ.
3. Modify each rule whose right-hand side has length greater than 1 and that contains a terminal symbol by substituting T_a for each occurrence of the terminal a.
4. Add to G, for each T_a, the rule $T_a \rightarrow a$.
5. Return G'.

Example:

$A \rightarrow$ a
$A \rightarrow$ a B
$A \rightarrow B$aC
$A \rightarrow B$bC

Long Rules

removeLong(G) =

1. Let $G' = G$.
2. For each rule r of the form:

$$A \rightarrow N_1N_2N_3N_4\ldots N_n,\ n > 2$$

create new nonterminals M_2, M_3, … $M_{n\text{-}1}$.

3. Replace r with the rule $A \rightarrow N_1M_2$.
4. Add the rules:

$M_2 \rightarrow N_2M_3$,
$M_3 \rightarrow N_3M_4$, …
$M_{n\text{-}1} \rightarrow N_{n\text{-}1}N_n$.

5. Return G'.

Example:
$A \rightarrow BCDEF$

An Example

$S \rightarrow aACa$
$A \rightarrow B \mid a$
$B \rightarrow C \mid c$
$C \rightarrow cC \mid \varepsilon$

removeEps returns:

$S \rightarrow aACa \mid aAa \mid aCa \mid aa$
$A \rightarrow B \mid a$
$B \rightarrow C \mid c$
$C \rightarrow cC \mid c$

An Example

$S \rightarrow aACa \mid aAa \mid aCa \mid aa$
$A \rightarrow B \mid a$
$B \rightarrow C \mid c$
$C \rightarrow cC \mid c$

Next we apply *removeUnits*:
Remove $A \rightarrow B$. Add $A \rightarrow C \mid c$.
Remove $B \rightarrow C$. Add $B \rightarrow cC$ ($B \rightarrow c$, already there).
Remove $A \rightarrow C$. Add $A \rightarrow cC$ ($A \rightarrow c$, already there).

So *removeUnits* returns:

$S \rightarrow aACa \mid aAa \mid aCa \mid aa$
$A \rightarrow a \mid c \mid cC$
$B \rightarrow c \mid cC$
$C \rightarrow cC \mid c$

An Example

$S \rightarrow aACa \mid aAa \mid aCa \mid aa$
$A \rightarrow a \mid c \mid cC$
$B \rightarrow c \mid cC$
$C \rightarrow cC \mid c$

Next we apply *removeMixed*, which returns:

$S \rightarrow T_aACT_a \mid T_aAT_a \mid T_aCT_a \mid T_aT_a$
$A \rightarrow a \mid c \mid T_cC$
$B \rightarrow c \mid T_cC$
$C \rightarrow T_cC \mid c$
$T_a \rightarrow a$
$T_c \rightarrow c$

An Example

$S \rightarrow T_aACT_a \mid T_aAT_a \mid T_aCT_a \mid T_aT_a$
$A \rightarrow a \mid c \mid T_cC$
$B \rightarrow c \mid T_cC$
$C \rightarrow T_cC \mid c$
$T_a \rightarrow a$
$T_c \rightarrow c$

Finally, we apply *removeLong*, which returns:

$S \rightarrow T_aS_1$ $\quad S \rightarrow T_aS_3$ $\quad S \rightarrow T_aS_4$ $\quad S \rightarrow T_aT_a$
$S_1 \rightarrow AS_2$ $\quad S_3 \rightarrow AT_a$ $\quad S_4 \rightarrow CT_a$
$S_2 \rightarrow CT_a$
$A \rightarrow a \mid c \mid T_cC$
$B \rightarrow c \mid T_cC$
$C \rightarrow T_cC \mid c$
$T_a \rightarrow a$
$T_c \rightarrow c$