# Parsing

## CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN

Chinese University of Hong Kong

Fall 2016

# Context-free versus regular

Write a CFG for the language $(0+1)^*111$

# Context-free versus regular

Write a CFG for the language $(0+1)^*111$

$$S \rightarrow U111$$
$$U \rightarrow 0U \mid 1U \mid \varepsilon$$

Can you do so for every regular language?

# Context-free versus regular

Write a CFG for the language $(0+1)^*111$

$$S \to U111$$
$$U \to 0U \mid 1U \mid \varepsilon$$

Can you do so for every regular language?

**Every regular language is context-free**

# From regular to context-free

| regular expression | $\Rightarrow$ CFG |
|---|---|
| $\varnothing$ | grammar with no rules |
| $\varepsilon$ | $S \to \varepsilon$ |
| a (alphabet symbol) | $S \to$ a |
| $E_1 + E_2$ | $S \to S_1 \mid S_2$ |
| $E_1 E_2$ | $S \to S_1 S_2$ |
| $E_1^*$ | $S \to S S_1 \mid \varepsilon$ |

$S$ becomes the new start variable

# Context-free versus regular

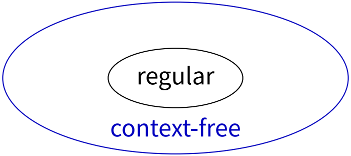
Is every context-free language regular?

# Context-free versus regular

Is every context-free language regular?

$S \to \mathtt{0}S\mathtt{1} \qquad L = \{\mathtt{0}^n\mathtt{1}^n \mid n \geqslant 0\}$

Is context-free but not regular

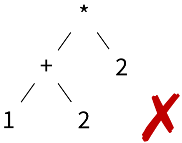


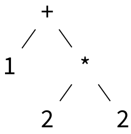
Ambiguity

# Ambiguity

$$E \to E\texttt{+}E \mid E\texttt{*}E \mid (E) \mid N$$
$$N \to 1N \mid 2N \mid 1 \mid 2$$

1+2*2




$= 6$ $= 5$

A CFG is ambiguous if some string has more than one parse tree

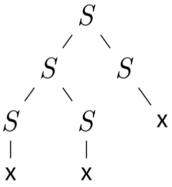
# Example

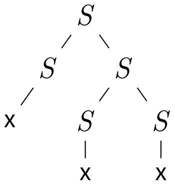
Is $\boxed{S \to SS|\text{x}}$ ambiguous?

# Example

Is $\boxed{S \rightarrow SS|\text{x}}$ ambiguous?

Yes, because

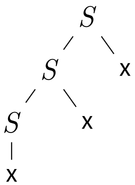



Two ways to derive xxx

# Disambiguation

$$\boxed{S \rightarrow SS|\mathsf{x}} \quad \Rightarrow \quad \boxed{S \rightarrow S\mathsf{x}|\mathsf{x}}$$



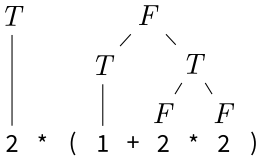

Sometimes we can rewrite the grammar to remove ambiguity

# Disambiguation

$$E \to E\texttt{+}E \mid E\texttt{*}E \mid (E) \mid N$$
$$N \to \texttt{1}N \mid \texttt{2}N \mid \texttt{1} \mid \texttt{2}$$

+ and * have the same precedence!

Divide expression into terms and factors



# Disambiguation

$$E \to E+E \mid E*E \mid (E) \mid N$$
$$N \to 1N \mid 2N \mid 1 \mid 2$$

An expression is a sum of one or more terms $\quad E \to T \mid E+T$

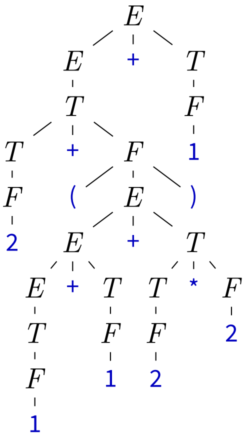
Each term is a product of one or more factors $\quad T \to F \mid T*F$

Each factor is a parenthesized expression or a number $\quad F \to (E) \mid 1 \mid 2$

# Parsing example

$$E \to T \mid E\texttt{+}T$$
$$T \to F \mid T\texttt{*}F$$
$$F \to (E) \mid \texttt{1} \mid \texttt{2}$$

Parse tree for
2+(1+1+2*2)+1



# Disambiguation

Disambiguation is not always possible because

There exists inherently ambiguous languages

There is no general procedure for disambiguation

# Disambiguation

Disambiguation is not always possible because
There exists inherently ambiguous languages
There is no general procedure for disambiguation

In programming languages, ambiguity comes from the precedence rules,
and we can resolve like in the example

In English, ambiguity is sometimes a problem:

I look at the dog with one eye

# Parsing

$S \to 0S1 \mid 1S0S \mid T$

$T \to S \mid \varepsilon$

input: 0011

Is $0011 \in L$?

If so, how to build a parse tree with a program?

# Parsing

$$S \to \texttt{0}S\texttt{1} \mid \texttt{1}S\texttt{0}S \mid T$$
$$T \to S \mid \varepsilon$$

input: 0011

Try all derivations?

$$S \to \texttt{0}S\texttt{1}$$
$$S \to \texttt{1}S\texttt{0}S$$
$$S \to T$$

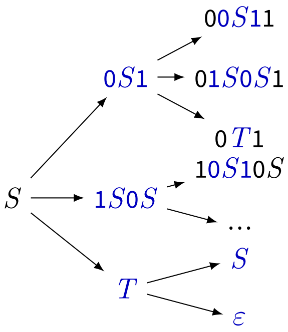
# Parsing

$S \to 0S1 \mid 1S0S \mid T$

$T \to S \mid \varepsilon$

input: 0011

Try all derivations?



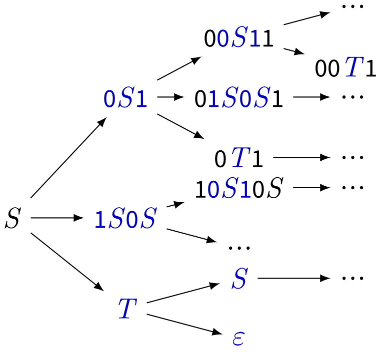
# Parsing

$S \to 0S1 \mid 1S0S \mid T$

$T \to S \mid \varepsilon$

input: 0011

Try all derivations?

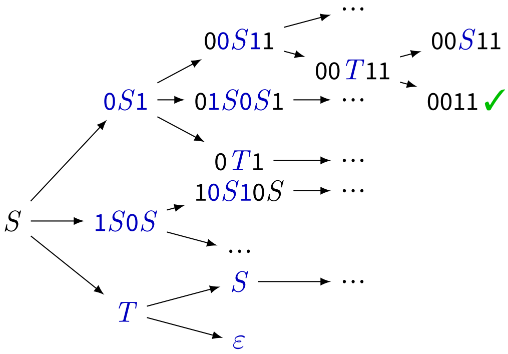


# Parsing

$$S \to 0S1 \mid 1S0S \mid T$$
$$T \to S \mid \varepsilon$$

input: 0011

Try all derivations?




This is (part of) the tree of all derivations, not the parse tree

# Problems

1. Trying all derivations may take too long
2. If input is not in the language, parsing will never stop

Let's tackle the 2nd problem

# When to stop

$$S \to \mathtt{0}S\mathtt{1} \mid \mathtt{1}S\mathtt{0}S \mid T$$
$$T \to \mathsf{S} \mid \varepsilon$$

Idea: Stop when
$|\text{derived string}| > |\text{input}|$

# When to stop

$$S \to 0S1 \mid 1S0S \mid T$$
$$T \to \mathsf{S} \mid \varepsilon$$

Idea: Stop when
$|\text{derived string}| > |\text{input}|$

Problems:

$$S \Rightarrow 0S1 \Rightarrow 0T1 \Rightarrow 01$$

Derived string may shrink
because of "$\varepsilon$-productions"

# When to stop

$$S \to 0S1 \mid 1S0S \mid T$$
$$T \to \mathsf{S} \mid \varepsilon$$

Idea: Stop when
$|\text{derived string}| > |\text{input}|$

Problems:

$$S \Rightarrow 0S1 \Rightarrow 0T1 \Rightarrow 01$$

Derived string may shrink
because of "$\varepsilon$-productions"

$$S \Rightarrow T \Rightarrow S \Rightarrow T \Rightarrow \ldots$$

Derviation may loop because
of "unit productions"

Remove $\varepsilon$ and unit productions

# Removing $\varepsilon$-productions

Goal: remove all $A \to \varepsilon$ rules for every non-start variable $A$

If $S$ is the start variable and the rule $S \to \varepsilon$ exists

Add a new start variable $T$

Add the rule $T \to S$

For every rule $A \to \varepsilon$ where $A$ is not the (new) start variable

1. Remove the rule $A \to \varepsilon$
2. If you see $B \to \alpha A \beta$
   Add a new rule $B \to \alpha\beta$

$$
\begin{aligned}
S &\to ACD \\
A &\to \mathsf{a} \\
B &\to \varepsilon \\
C &\to ED \mid \varepsilon \\
D &\to BC \mid \mathsf{b} \\
E &\to \mathsf{b}
\end{aligned}
$$

# Removing $\varepsilon$-productions

Goal: remove all $A \to \varepsilon$ rules for every non-start variable $A$

If $S$ is the start variable and the rule $S \to \varepsilon$ exists

Add a new start variable $T$
Add the rule $T \to S$

For every rule $A \to \varepsilon$ where $A$ is not the (new) start variable

1. Remove the rule $A \to \varepsilon$
2. If you see $B \to \alpha A \beta$
   Add a new rule $B \to \alpha\beta$

$$
\begin{aligned}
S &\to ACD \\
A &\to \mathrm{a} \\
\cancel{B} &\cancel{\to \varepsilon} \\
C &\to ED \mid \varepsilon \\
D &\to BC \mid \mathrm{b} \\
E &\to \mathrm{b}
\end{aligned}
$$

$$D \to C$$

Removing $B \to \varepsilon$

# Removing $\varepsilon$-productions

Goal: remove all $A \to \varepsilon$ rules for every non-start variable $A$

If $S$ is the start variable and the rule $S \to \varepsilon$ exists

Add a new start variable $T$
Add the rule $T \to S$

For every rule $A \to \varepsilon$ where $A$ is not the (new) start variable

1. Remove the rule $A \to \varepsilon$
2. If you see $B \to \alpha A \beta$
   Add a new rule $B \to \alpha\beta$

$$
\begin{aligned}
S &\to ACD \\
A &\to \mathrm{a} \\
&\cancel{B \to \varepsilon} \\
C &\to ED \mid \cancel{\varepsilon} \\
D &\to BC \mid \mathrm{b} \\
E &\to \mathrm{b}
\end{aligned}
$$

$$
\begin{aligned}
D &\to C \\
S &\to AD
\end{aligned}
$$

Removing $C \to \varepsilon$

# Removing $\varepsilon$-productions

Goal: remove all $A \to \varepsilon$ rules for every non-start variable $A$

If $S$ is the start variable and the rule $S \to \varepsilon$ exists

Add a new start variable $T$
Add the rule $T \to S$

For every rule $A \to \varepsilon$ where $A$ is not the (new) start variable

1. Remove the rule $A \to \varepsilon$
2. If you see $B \to \alpha A \beta$
   Add a new rule $B \to \alpha\beta$

$$
\begin{aligned}
S &\to ACD \\
A &\to \text{a} \\
&\cancel{B \to \varepsilon} \\
C &\to ED \mid \cancel{\varepsilon} \\
D &\to BC \mid \text{b} \\
E &\to \text{b}
\end{aligned}
$$

$$
\begin{aligned}
D &\to C \\
S &\to AD \\
D &\to \varepsilon
\end{aligned}
$$

Removing $C \to \varepsilon$

# Removing $\varepsilon$-productions

Goal: remove all $A \to \varepsilon$ rules for every non-start variable $A$

If $S$ is the start variable and the rule $S \to \varepsilon$ exists

Add a new start variable $T$
Add the rule $T \to S$

For every rule $A \to \varepsilon$ where $A$ is not the (new) start variable

1. Remove the rule $A \to \varepsilon$
2. If you see $B \to \alpha A \beta$
   Add a new rule $B \to \alpha\beta$

$$
\begin{aligned}
S &\to ACD \\
A &\to \text{a} \\
\cancel{B} &\cancel{\to \varepsilon} \\
C &\to ED \mid \cancel{\varepsilon} \\
D &\to BC \mid \text{b} \\
E &\to \text{b}
\end{aligned}
$$

$$
\begin{aligned}
D &\to C \\
S &\to AD \\
\cancel{D} &\cancel{\to \varepsilon} \\
C &\to E
\end{aligned}
$$

Removing $D \to \varepsilon$

# Removing $\varepsilon$-productions

Goal: remove all $A \to \varepsilon$ rules for every non-start variable $A$

If $S$ is the start variable and the rule $S \to \varepsilon$ exists

Add a new start variable $T$
Add the rule $T \to S$

For every rule $A \to \varepsilon$ where $A$ is not the (new) start variable

1. Remove the rule $A \to \varepsilon$
2. If you see $B \to \alpha A \beta$
   Add a new rule $B \to \alpha\beta$

$$
\begin{aligned}
S &\to ACD \\
A &\to \text{a} \\
\cancel{B} &\cancel{\to \varepsilon} \\
C &\to ED \mid \cancel{\varepsilon} \\
D &\to BC \mid \text{b} \\
E &\to \text{b}
\end{aligned}
$$

$$
\begin{aligned}
D &\to C \\
S &\to AD \\
\cancel{D} &\cancel{\to \varepsilon} \\
C &\to E \\
S &\to A
\end{aligned}
$$

Removing $D \to \varepsilon$

# Eliminating $\varepsilon$-productions

For every $A \to \varepsilon$ rule where $A$ is not the start variable

1. Remove the rule $A \to \varepsilon$
2. If you see $B \to \alpha A \beta$
   Add a new rule $B \to \alpha\beta$

Do 2. every time $A$ appears

$B \to \alpha A \beta A \gamma$ yields

$$B \to \alpha\beta A\gamma \quad B \to \alpha A \beta\gamma$$
$$B \to \alpha\beta\gamma$$

# Eliminating $\varepsilon$-productions

For every $A \to \varepsilon$ rule where $A$ is not the start variable

1. Remove the rule $A \to \varepsilon$
2. If you see $B \to \alpha A \beta$
   Add a new rule $B \to \alpha\beta$

Do 2. every time $A$ appears

$B \to \alpha A \beta A \gamma$ yields

$$B \to \alpha\beta A\gamma \quad B \to \alpha A\beta\gamma$$
$$B \to \alpha\beta\gamma$$

$B \to A$ becomes $B \to \varepsilon$

If $B \to \varepsilon$ was removed earlier, don't add it back

# Eliminating unit productions

A unit production is a production of the form

$$A \to B$$

Grammar:

$$\begin{aligned} S &\to 0S1 \mid 1S0S \mid T \\ T &\to S \mid R \mid \varepsilon \\ R &\to 0SR \end{aligned}$$

Unit production graph:

$$\begin{tikzcd} S \arrow[r, bend left] & T \arrow[l, bend left] \arrow[d] \\ & R \end{tikzcd}$$

# Removing unit productions

① If there is a cycle of unit productions

$$A \to B \to \cdots \to C \to A$$

delete it and replace everything with $A$

$$S \to \mathtt{0}S\mathtt{1} \mid \mathtt{1}S\mathtt{0}S \mid T$$
$$T \to S \mid R \mid \varepsilon$$
$$R \to \mathtt{0}SR$$

$$\begin{tikzcd} S \arrow[r, bend left] & T \arrow[l, bend left] \arrow[d] \\ & R \end{tikzcd}$$

# Removing unit productions

① If there is a cycle of unit productions

$$A \to B \to \cdots \to C \to A$$

delete it and replace everything with $A$

$$\begin{aligned} S &\to \mathtt{0}S\mathtt{1} \mid \mathtt{1}S\mathtt{0}S \mid \cancel{T} \\ \cancel{T} &\to \cancel{S} \mid R \mid \varepsilon \\ R &\to \mathtt{0}SR \end{aligned}$$

$$\begin{tikzcd} S \arrow[r, bend left] & T \arrow[l, bend left] \arrow[d] \\ & R \end{tikzcd}$$

$$\begin{aligned} S &\to \mathtt{0}S\mathtt{1} \mid \mathtt{1}S\mathtt{0}S \\ S &\to R \mid \varepsilon \\ R &\to \mathtt{0}SR \end{aligned}$$

Replace $T$ by $S$

# Removal of unit productions

② replace any chain

$$A \to B \to \cdots \to C \to \alpha$$

$$\text{by} \quad A \to \alpha, \quad B \to \alpha, \quad \ldots, \quad C \to \alpha$$

$$\begin{aligned} S &\to \mathtt{0}S\mathtt{1} \mid \mathtt{1}S\mathtt{0}S \\ &\quad \mid R \mid \varepsilon \\ R &\to \mathtt{0}SR \end{aligned}$$

$$\begin{tikzcd} S \arrow[d] \\ R \end{tikzcd}$$

# Removal of unit productions

② replace any chain

$$A \to B \to \cdots \to C \to \alpha$$

by $\quad A \to \alpha, \quad B \to \alpha, \quad \ldots, \quad C \to \alpha$

$$\begin{aligned} S &\to \mathtt{0}S\mathtt{1} \mid \mathtt{1}S\mathtt{0}S \\ &\mid R \mid \varepsilon \\ R &\to \mathtt{0}SR \end{aligned} \qquad\qquad S \downarrow R \qquad\qquad \begin{aligned} S &\to \mathtt{0}S\mathtt{1} \mid \mathtt{1}S\mathtt{0}S \\ &\mid \mathtt{0}SR \mid \varepsilon \\ R &\to \mathtt{0}SR \end{aligned}$$

Replace $\quad S \to R \to \mathtt{0}SR \quad$ by $\quad S \to \mathtt{0}SR, \quad R \to \mathtt{0}SR$

# Recap

Problems:

1. Trying all derivations may take too long
2. If input is not in the language, parsing will never stop ✓

Solution to problem 2:

1. Eliminate $\varepsilon$ productions
2. Eliminate unit productions

Try all possible derivations but stop parsing when
$|\text{derived string}| > |\text{input}|$

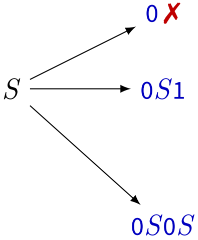
# Example

$$S \to 0S1 \mid 0S0S \mid T$$
$$T \to S \mid 0$$

$$\Longrightarrow$$

$$S \to 0S1 \mid 0S0S \mid 0$$

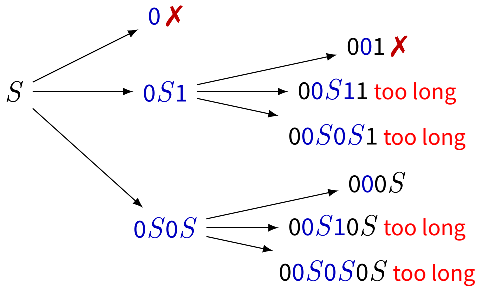
input: 0011



# Example

$$S \to 0S1 \mid 0S0S \mid T \qquad\qquad \Longrightarrow \qquad\qquad S \to 0S1 \mid 0S0S \mid 0$$
$$T \to S \mid 0$$

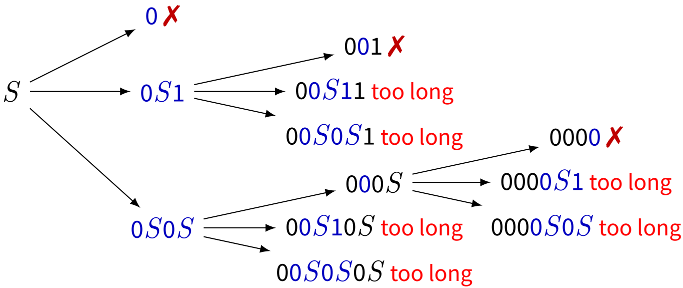
input: 0011



# Example

$$S \to 0S1 \mid 0S0S \mid T \qquad\qquad S \to 0S1 \mid 0S0S \mid 0$$
$$T \to S \mid 0 \qquad \Longrightarrow$$

input: 0011




Conclusion: $0011 \notin L$

# Problems

1. Trying all derivations may take too long
2. If input is not in the language, parsing will never stop

# Preparations

A faster way to parse:

Cocke–Younger–Kasami algorithm

To use it we must perprocess the CFG:

Eliminate $\varepsilon$ productions
Eliminate unit productions
Convert CFG to Chomsky Normal Form

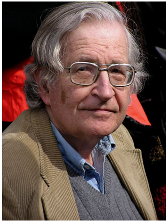
# Chomsky Normal Form

A CFG is in Chomsky Normal Form if every production has the form

$A \to BC$ or $A \to \text{a}$

where neither $B$ nor $C$ is the start variable

but we also allow $S \to \varepsilon$ for start variable $S$



Noam Chomsky

Convert to Chomsky Normal Form:

$A \to B\text{c}DE$ $\Longrightarrow$ replace terminals with new variables $A \to BCDE$, $C \to \text{c}$ $\Longrightarrow$ break up sequences with new variables $A \to BX$, $X \to CY$, $Y \to DE$, $C \to \text{c}$

# Cocke–Younger–Kasami algorithm

$$S \to AB \mid BC$$
$$A \to BA \mid \mathsf{a}$$
$$B \to CC \mid \mathsf{b}$$
$$C \to AB \mid \mathsf{a}$$

Input: $x =$ baaba

let
$x[i, \ell] = x_i x_{i+1} \dots x_{i+\ell-1}$

| $\ell$ | | | | | | |
|---|---|---|---|---|---|---|
| 5 | | | | | | |
| 4 | | | | | | |
| 3 | | | | | | |
| 2 | | | | | | |
| 1 | | | | | | |
| | 1 | 2 | 3 | 4 | 5 | $i$ |
| | b | a | a | b | a | |

For every substring $x[i, \ell]$, remember all variables $R$ that derive $x[i, \ell]$
Store in a table $T[i, \ell]$

# Cocke–Younger–Kasami algorithm

$$S \to AB \mid BC$$
$$A \to BA \mid \mathsf{a}$$
$$B \to CC \mid \mathsf{b}$$
$$C \to AB \mid \mathsf{a}$$

Input: $x =$ baaba

let
$x[i,\ell] = x_i x_{i+1} \dots x_{i+\ell-1}$

| $\ell$ | | | | | |
|---|---|---|---|---|---|
| 5 | | | | | |
| 4 | | | | | |
| 3 | | | | | |
| 2 | | | | | |
| 1 | $B$ | | | | |
| | 1 | 2 | 3 | 4 | 5 |
| | b | a | a | b | a |

For every substring $x[i,\ell]$, remember all variables $R$ that derive $x[i,\ell]$
Store in a table $T[i,\ell]$

# Cocke–Younger–Kasami algorithm

$$S \to AB \mid BC$$
$$A \to BA \mid \mathsf{a}$$
$$B \to CC \mid \mathsf{b}$$
$$C \to AB \mid \mathsf{a}$$

Input: $x =$ baaba

let
$x[i, \ell] = x_i x_{i+1} \ldots x_{i+\ell-1}$

| $\ell$ | | | | | | |
|---|---|---|---|---|---|---|
| 5 | | | | | | |
| 4 | | | | | | |
| 3 | | | | | | |
| 2 | | | | | | |
| 1 | $B$ | $A \mid C$ | $A \mid C$ | $B$ | $A \mid C$ | |
| | 1 | 2 | 3 | 4 | 5 | $i$ |
| | b | a | a | b | a | |

For every substring $x[i, \ell]$, remember all variables $R$ that derive $x[i, \ell]$
Store in a table $T[i, \ell]$

# Cocke–Younger–Kasami algorithm

$$S \to AB \mid BC$$
$$A \to BA \mid \mathsf{a}$$
$$B \to CC \mid \mathsf{b}$$
$$C \to AB \mid \mathsf{a}$$

Input: $x =$ baaba

let
$x[i, \ell] = x_i x_{i+1} \dots x_{i+\ell-1}$

| $\ell$ | | | | | | |
|---|---|---|---|---|---|---|
| 5 | | | | | | |
| 4 | | | | | | |
| 3 | | | | | | |
| 2 | $S \vert A$ | | | | | |
| 1 | $B$ | $A \vert C$ | $A \vert C$ | $B$ | $A \vert C$ | |
| | 1 | 2 | 3 | 4 | 5 | $i$ |
| | b | a | a | b | a | |

For every substring $x[i, \ell]$, remember all variables $R$ that derive $x[i, \ell]$
Store in a table $T[i, \ell]$

# Cocke–Younger–Kasami algorithm

$$S \to AB \mid BC$$
$$A \to BA \mid \mathsf{a}$$
$$B \to CC \mid \mathsf{b}$$
$$C \to AB \mid \mathsf{a}$$

Input: $x =$ baaba

let
$x[i, \ell] = x_i x_{i+1} \dots x_{i+\ell-1}$

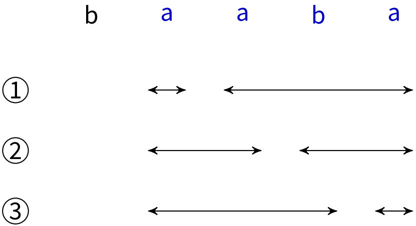
| $\ell$ | | | | | | |
|---|---|---|---|---|---|---|
| 5 | | | | | | |
| 4 | | | | | | |
| 3 | | | | | | |
| 2 | $S \mid A$ | $B$ | $S \mid C$ | $S \mid A$ | | |
| 1 | $B$ | $A \mid C$ | $A \mid C$ | $B$ | $A \mid C$ | |
| | 1 | 2 | 3 | 4 | 5 | $i$ |
| | b | a | a | b | a | |

For every substring $x[i, \ell]$, remember all variables $R$ that derive $x[i, \ell]$
Store in a table $T[i, \ell]$

# Computing $T[i,\ell]$ for $\ell \geqslant 2$

To compute $T[2,4]$

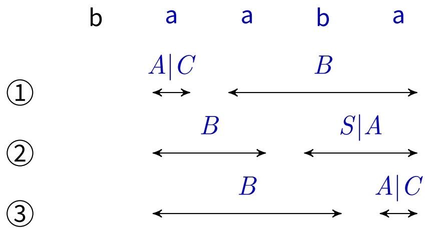
Try all possible ways to split $x[2,4]$ into two substrings



# Computing $T[i, \ell]$ for $\ell \geqslant 2$

To compute $T[2, 4]$

Try all possible ways to split $x[2, 4]$ into two substrings

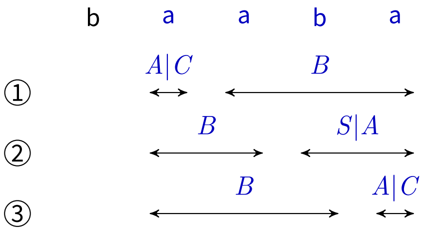



Look up entries regarding shorter substrings previously computed

# Computing $T[i, \ell]$ for $\ell \geqslant 2$

To compute $T[2,4]$

Try all possible ways to split $x[2,4]$ into two substrings




Look up entries regarding shorter substrings previously computed

$$S \to AB \mid BC$$
$$A \to BA \mid \text{a}$$
$$B \to CC \mid \text{b}$$
$$C \to AB \mid \text{a}$$

$$T[2,4] = S|A|C$$

# Cocke–Younger–Kasami algorithm

$$S \to AB \mid BC$$
$$A \to BA \mid \mathsf{a}$$
$$B \to CC \mid \mathsf{b}$$
$$C \to AB \mid \mathsf{a}$$

Input: $x = $ baaba

| $\ell$ | | | | | |
|---|---|---|---|---|---|
| 5 | $S\|A\|C$ | | | | |
| 4 | - | $S\|A\|C$ | | | |
| 3 | - | $B$ | $B$ | | |
| 2 | $S\|A$ | $B$ | $S\|C$ | $S\|A$ | |
| 1 | $B$ | $A\|C$ | $A\|C$ | $B$ | $A\|C$ |
| | 1 | 2 | 3 | 4 | 5 |
| | b | a | a | b | a |

$i$

Get parse tree by tracing back derivations