Bab 7 Sifat-Sifat Bahasa Context-Free

Konversi Context- Free Grammars ke Chomsky Normal Form (CFG ke CNF)

Informatika

Chomsky Normal Form

- **A CFG is said to be in *Chomsky Normal Form* if every production is of one of these two forms:**
  1. A -> BC (right side is two variables).
  2. A -> a (right side is a single terminal).
- **Theorem: If L is a CFL, then L – {ϵ} has a CFG in CNF.**

Mengubah CFG ke CNF

- **Langkah:**
  1. Hapus ϵ-productions
  2. Hapus unit productions
     1. ~~A → B~~
     2. B → 0 | CD
     3. A → 0 | CD
  3. Hapus useless variable
     1. Yang tidak menurunkan string terminal
     2. Yang tidak bisa dicapai dari start symbol
  4. Ubah ke bentuk
     1. A -> BC (bagian kanan terdiri dari 2 variable)
     2. A -> a (bagian kanan adalah single terminal)

Menghapus Non Terminal (Variable) yg Useless

- **Useless:**
  - NT tidak bisa diturunkan sampai ke terminal
    - Tidak ada penurunan yang akhirnya sampai ke NT * terminal
  - NT tidak bisa dicapai dari start symbol $\rightarrow$

Contoh: NT yg tdk bisa diturunkan sampai terminal

S -> AB | C, A -> aA | a, B -> bB, C -> c

- **Basis: A dan C teridentifikasi berguna karena A -> a dan C -> c.**
- **Induction: S teridentifikasi karena S -> C sehingga S -*-> c**
- **Yang lainnya tidak bisa teridentifikasi yaitu B sehingga**
  - Hasil: S -> C, A -> aA | a, C -> c

Menghapus ϵ-production

- **Utk menghapus ϵ-production, pertama definisikan *nullable variables***
  - Variable yg bisa diturunkan menjadi string kosong yaitu A -->* ϵ
- **Basis: jika ada A -> ϵ, maka A adalah nullable.**
- **Induction: Jika ada A -> α, dan semua symbol α adalah nullable, maka A adalah nullable.**
- **Jika A -> $X_1 \ldots X_n$ into a family of productions, dan ada k≥1, 1≤ k≤n dari X adalah symbol nullable maka aturan produksi baru semua kombinasi X .**

S → Ba
B → eps | b

S → Ba | a
B → b

Contoh: Nullable Symbols

S -> AB, A -> aA | ϵ, B -> bB | A

- **Basis: A adalah nullable karena A -> ϵ**
- **Induction: B adalah nullable karena B -> A.**
- **Maka S adalah nullable karena S -> AB.**

Contoh: Menghapus ϵ-Productions

S -> ABC, A -> aA | ϵ, B -> bB | ϵ, C -> ϵ

- **A, B, C, dan S adalah nullable**
  - C -> ϵ tidak bisa diganti krn tdk ada produksi lain
  - Aturan produksi utk B diganti jadi B -> bB | b
  - Aturan produksi utk A diganti jadi A -> aA | a
  - Aturan produksi utk S diganti jadi S -> ABC | AB | AC | BC | A | B | C
- **Karena aturan produksi utk C hilang maka**
  - Aturan produksi utk S jadi S -> AB | A | B
- **Hasilnya :** S -> AB | A | B, A -> aA | a, B -> bB | b

Menghapus Unit Productions

- ***Unit production*** **adalah aturan produksi yang bagian kanannya hanya terdiri dari satu variable, contoh A -> B**
- **Cara:**
  - Jika A =>* B adalah unit production, dan B -> α adalah non-unit-production, maka masukkan A -> α
  - Kemudian hapus semua unit productions

Let's look at grammar

$$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$
$$F \to I \mid (E)$$
$$T \to F \mid T * F$$
$$E \to T \mid E + T$$

It has unit productions $E \to T$, $T \to F$, and $F \to I$

We'll expand rule $E \to T$ and get rules

$$E \to F,\ E \to T * F$$

We then expand $E \to F$ and get

$$E \to I|(E)|T * F$$

Finally we expand $E \to I$ and get

$$E \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E) \mid T * F$$

Contoh CFG ke CNF (Ex 7.1.2)

- S → ASB | ϵ
- A → aAS | a
- B → SbS | A | bb
- menghilangkan ϵ production
  - S → ASB | AB
  - A → aAS | a | aA
  - B → SbS | A | bb | bS | Sb | b
- menghilangkan unit production
  - B → SbS | aAS | a | aA | bb | bS | Sb | b

Lanjutan Contoh CFG ke CNF

- S → ASB | AB
- A → aAS | a | aA
- B → SbS | aAS | a | aA | bb | bS | Sb | b

- **membuat bagian kanan menjadi 2 non terminal atau 1 terminal**
  - S → AC | AB
  - C → SB
  - A → DE | a | DA
  - D → a
  - E → AS
  - B → SF | DE | a | DA | GG | GS | SG | b
  - G → b
  - F → GS

CLOSURE PROPERTIES CFL

Closure Properties CFL

- **Closure properties=beberapa operasi dlm CFL yang pasti menghasilkan CFL:**
  - Theorema Substitution:
    - Union
    - Concatenation
    - Star Closure
    - Homomorphism
  - Reversal
- **Yg bukan Closure properties CFL:**
  - Intersection antar CFL
  - Complementation dari sebuah CFL

Union

- **Ada G1 dan G2 dimana**
  - G1 = {V1,T1,P1,S1}
  - G2 = {V2,T2,P2,S2}

- **Maka G = G1 U G2 adalah**
  - V baru = V1 ∪ V2 ∪ {S}
    - S adalah start symbol baru (tambahan baru)
  - P baru = P1 ∪ P2 ∪ {S→ S1 | S2}

Theorem: CFLs are closed under union

If L_1 and L_2 are CFLs, then $L_1 \cup L_2$ is a CFL.

Proof

1. Let L_1 and L_2 be generated by the CFG, $G_1 = (V_1, T_1, P_1, S_1)$ and $G_2 = (V_2, T_2, P_2, S_2)$, respectively.
2. Without loss of generality, subscript each nonterminal of G_1 with a 1, and each nonterminal of G_2 with a 2 (so that $V_1 \cap V_2 = \emptyset$).
3. Define the CFG, G, that generates $L_1 \cup L_2$ as follows:
   $G = (V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \rightarrow S_1 \mid S_2\}, S)$.
4. A derivation starts with either $S \Rightarrow S_1$ or $S \Rightarrow S_2$.
5. Subsequent steps use productions entirely from G_1 or entirely from G_2.
6. Each word generated thus is either a word in L_1 or a word in L_2.

Claim 1.1.1 *The class of CFLs is closed under the union ($\cup$) operation.*

Proof Idea: We need to pick up *any* two CFLs, say L_1 and L_2 and then show that the union of these languages, $L_1 \cup L_2$ is a CFL. But how do we show that a language is a context free? One method for this is to come up with a CFG and show that the grammar generates the language. We know that since L_1 and L_2 are CFLs, we have some CFGs say G_1 and G_2 that generate L_1 and L_2 respectively. So we may try to use these two grammars to construct the grammar that generates $L_1 \cup L_2$. In the exam you need not write the proof idea. This is just for conveying the idea. You need to write the proof as below.

Proof: Let L_1, L_2 be any two CFL, we will show that $L = L_1 \cup L_2$ is a CFL. Since L_1, L_2 are CFLs, there must exist CFGs which generate these two languages. Let G_1 and G_2 generate the languages L_1 and L_2 respectively, where:

$$G_1 = (V_1, \Sigma_1, R_1, S_1), \text{ and}$$

$$G_2 = (V_2, \Sigma_2, R_2, S_2)$$

We assume that the sets V_1 and V_2 are disjoint, or $V_1 \cap V_2 = \phi$ (we can always assume this because if the sets are not disjoint we can make them so, by renaming variables in one of the grammars). Consider the following grammar:

$$G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \rightarrow S_1 | S_2\}, S)$$

The above grammar is basically a combination of the grammars G_1 and G_2 in which we have added the new start state S and a new production rule $S \rightarrow S_1|S_2$. Now we need to show that G generates L(this is basically the proof of correctness of the construction). For this we need to show the following two things:

1. For any string $s \in L$, G generates s: We know that either $s \in L_1$ or $s \in L_2$ which implies that either $S_1 \Rightarrow^* s$ or $S_2 \Rightarrow^* s$. Since G has the production $S \rightarrow S_1|S_2$ we can conclude that $S \Rightarrow^* s$. So G generates s.

2. Let s be any string generated by G, then $s \in L$: We have $S \Rightarrow^* s$, this means that either $S_1 \Rightarrow^* s$ or $S_2 \Rightarrow^* s$. Now since we have made sure that $V_1 \cap V_2 = \phi$, s is either derived from S_1 using the rules R_1 only or it is derived from S_2 using rules R_2 only. This means that $s \in L_1 \cup L_2$.

Example

- Let L_1 be PALINDROME, defined by:

  $S \rightarrow aSa \mid bSb \mid a \mid b \mid \Lambda$

- Let L_2 be $\{a^n b^n | n \geq 0\}$ defined by:

  $S \rightarrow aSb \mid \Lambda$

- Then the union language is defined by:

  $S \rightarrow S_1 \mid S_2$

  $S_1 \rightarrow aS_1a \mid bS_1b \mid a \mid b \mid \Lambda$

  $S_2 \rightarrow aS_2b \mid \Lambda$

Concatenation

- **Ada G1 dan G2 dimana**
  - G1 = {V1,T1,P1,S1}
  - G2 = {V2,T2,P2,S2}
- **Maka L = {wv|w ϵ L(G1) dan v ϵ L(G2)} adalah**
  - V baru = V1 ∪ V2 ∪ {S}
    - S adalah start symbol baru (tambahan baru)
  - P baru = P1 ∪ P2 ∪ {S→ S1 S2}

Theorem: CFLs are closed under concatenation

If L_1 and L_2 are CFLs, then L_1L_2 is a CFL.

Proof

1. Let L_1 and L_2 be generated by the CFG, $G_1 = (V_1, T_1, P_1, S_1)$ and $G_2 = (V_2, T_2, P_2, S_2)$, respectively.
2. Without loss of generality, subscript each nonterminal of G_1 with a 1, and each nonterminal of G_2 with a 2 (so that $V_1 \cap V_2 = \emptyset$).
3. Define the CFG, G, that generates L_1L_2 as follows:
   $G = (V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \rightarrow S_1S_2\}, S)$.
4. Each word generated thus is a word in L_1 followed by a word in L_2.

Example

- Let L_1 be PALINDROME, defined by:

  $S \rightarrow aSa \mid bSb \mid a \mid b \mid \Lambda$

- Let L_2 be $\{a^n b^n | n \geq 0\}$ defined by:

  $S \rightarrow aSb \mid \Lambda$

- Then the concatenation language is defined by:

  $S \rightarrow S_1 S_2$

  $S_1 \rightarrow aS_1a \mid bS_1b \mid a \mid b \mid \Lambda$

  $S_2 \rightarrow aS_2b \mid \Lambda$

Star Closure

- **Ada G1 dimana**
  - G1 = {V1,T1,P1,S1}
- **Maka L baru = {w|w ϵ L(G1)* } adalah**
  - V baru = V1 ∪ {S}
    - S adalah start symbol baru (tambahan baru)
  - P baru = P1 ∪ {S→ S1 S| ϵ }

Theorem: CFLs are closed under Kleene star

If L_1 is a CFL, then L_1^* is a CFL.

Proof

1. Let L_1 be generated by the CFG, $G_1 = (V_1, T_1, P_1, S_1)$.
2. Without loss of generality, subscript each nonterminal of G_1 with a 1.
3. Define the CFG, G, that generates L_1^* as follows:
   $G = (V_1 \cup \{S\}, T_1, P_1 \cup \{S \rightarrow S_1 S \mid \Lambda\}, S)$.
4. Each word generated is either Λ or some sequence of words in L_1.
5. Every word in L_1^* (i.e., some sequence of 0 or more words in L_1) can be generated by G.

Example

- Let L_1 be $\{a^n b^n | n \geq 0\}$ defined by:

  $S \rightarrow aSb \mid \Lambda$

- Then L_1^* is generated by:

  $S \rightarrow S_1 S \mid \Lambda$

  $S_1 \rightarrow aS_1 b \mid \Lambda$

None of these example grammars is necessarily the most *compact* CFG for the language it generates.

Homomorphisms

- **A *homomorphism* on an alphabet is a function that gives a string for each symbol in that alphabet.**
- **Example: h(0) = ab; h(1) = ϵ.**
- **Extend to strings by $h(a_1 \ldots a_n) = h(a_1) \ldots h(a_n)$.**
- **Example: h(01010) = abababab.**

Homomorphism

Proposition 8. *Context free languages are closed under homomorphisms.*

Proof. Let $G = (V, \Sigma, R, S)$ be the grammar generating L, and let $h : \Sigma^* \to \Gamma^*$ be a homomorphism. A grammar $G' = (V', \Gamma, R', S')$ for generating $h(L)$:

- Include all variables from G (i.e., $V' \supseteq V$), and let $S' = S$
- Treat terminals in G as variables. i.e., for every $a \in \Sigma$
  - Add a new variable X_a to V'
  - In each rule of G, if a appears in the RHS, replace it by X_a
- For each X_a, add the rule $X_a \to h(a)$

G' generates $h(L)$. *(Exercise!)* □

Example 9. Let G have the rules $S \to 0S0|1S1|\epsilon$.
Consider the homorphism $h : \{0,1\}^* \to \{a,b\}^*$ given by $h(0) = aba$ and $h(1) = bb$.
Rules of G' s.t. $\mathbf{L}(G') = \mathbf{L}(L(G))$:

$$\begin{aligned} S &\to X_0SX_0|X_1SX_1|\epsilon \\ X_0 &\to aba \\ X_1 &\to bb \end{aligned}$$

Inverse Homomorphisms

Recall: For a homomorphism h, $h^{-1}(L) = \{w \mid h(w) \in L\}$

Proposition 10. *If L is a CFL then $h^{-1}(L)$ is a CFL*

Proof Idea
For regular language L: the DFA for $h^{-1}(L)$ on reading a symbol a, simulated the DFA for L on $h(a)$. Can we do the same with PDAs?

- Key idea: store $h(a)$ in a "buffer" and process symbols from $h(a)$ one at a time (according to the transition function of the original PDA), and the next input symbol is processed only after the "buffer" has been emptied.
- Where to store this "buffer"? In the state of the new PDA!

Proof. Let $P = (Q, \Delta, \Gamma, \delta, q_0, F)$ be a PDA such that $\mathbf{L}(P) = L$. Let $h : \Sigma^* \to \Delta^*$ be a homomorphism such that $n = \max_{a \in \Sigma} |h(a)|$, i.e., every symbol of Σ is mapped to a string under h of length at most n. Consider the PDA $P' = (Q', \Sigma, \Gamma, \delta', q_0', F')$ where

- $Q' = Q \times \Delta^{\leq n}$, where $\Delta^{\leq n}$ is the collection of all strings of length at most n over Δ.
- $q_0' = (q_0, \epsilon)$
- $F' = F \times \{\epsilon\}$
- δ' is given by

$$\delta'((q, v), x, a) = \begin{cases} \{((q, h(x)), \epsilon)\} & \text{if } v = a = \epsilon \\ \{((p, u), b) \mid (p, b) \in \delta(q, y, a)\} & \text{if } v = yu,\ x = \epsilon, \text{ and } y \in (\Delta \cup \{\epsilon\}) \end{cases}$$

and $\delta'(\cdot) = \emptyset$ in all other cases.

We can show by induction that for every $w \in \Sigma^*$

$$\langle q_0', \epsilon \rangle \xrightarrow{w}_{P'} \langle (q, v), \sigma \rangle \text{ iff } \langle q_0, \epsilon \rangle \xrightarrow{w'}_{P} \langle q, \sigma \rangle$$

where $h(w) = w'v$. Again this induction proof is left as an exercise. Now, $w \in \mathbf{L}(P')$ iff $\langle q_0', \epsilon \rangle \xrightarrow{w}_{P'}$ $\langle (q, \epsilon), \sigma \rangle$ where $q \in F$ (by definition of PDA acceptance and F') iff $\langle q_0, \epsilon \rangle \xrightarrow{h(w)}_{P} \langle q, \sigma \rangle$ (by exercise) iff $h(w) \in \mathbf{L}(P)$ (by definition of PDA acceptance). Thus, $\mathbf{L}(P') = h^{-1}(\mathbf{L}(P)) = h^{-1}(L)$. □

Reversal

- **Ada G dimana**
  - G = {V, T, P, S}
- **Maka L baru = {w|w ϵ $L(G)^R$ } adalah**
  - P baru adalah jika {S→ α } suatu aturan produksi di P maka P baru menjadi S→ α^R

Exercise 7.3.1

Show that the operation *cycle* preserves context-free languages, where cycle is defined by:

$$cycle(L) = \{xy \mid yx \in L\}$$

Informally, $cycle(L)$ allows all strings constructed as follows: take a string w from L, and choose an arbitrary position in the middle. Take the second part (this is x), then wrap around and concatenate the first part (this is y). For example, if $abcd \in L$, then all of the following strings are in $cycle(L)$: $abcd, bcda, cdab, dabc$.

Solution

We will prove this by PDA construction. Given a PDA P for L, we'll create PDA P_c for $cycle(L)$. Both accept by empty stack. P_c simulates an accepting computation in P, but it starts in the middle and simulates P on x, then "wraps around" and simulates P on y.

The difficulty lies in the stack. When P_c begins consuming x, it is missing the stack symbols that P would have from consuming y. To resolve this, we allow P_c to guess the top stack symbol and "pop" it by pushing a negated version. Later, when simulating P on y, we confirm these guesses: instead of pushing symbols, we confirm and pop the guesses.

We use a stack marker # to delimit the guessed symbols from the rest of the stack. Start by guessing the top symbol (the first one we'll pop) left by y. Place a copy above and below the marker. Then begin consuming x by simulating P normally on the portion above the stack above the marker. Eventually we return to the marker, meaning we popped the symbol we guessed (note: it is now recorded below the marker). Here is an example where we guess that y left A as the top stack symbol:

Start configuration	Z_0
Mark the stack	$\#Z_0$
Guess top symbol	$A\#AZ_0$
Arbitrary build up	$DEFA\#AZ_0$
	...
Eventually pop A	$\#AZ_0$

On return to the marker, one thing we can do is guess the next symbol to pop and repeat the step described above:

Guess B	$B\#BAZ_0$
...	...
Eventually pop B	$\#BAZ_0$
Guess C	$C\#CBAZ_0$
...	...
Eventually pop C	$\#CBAZ_0$

Another option when returning to the marker is to guess that we have finished consuming x – this means we must have emptied the stack left by y, so it is time to start consuming y and verifying that it builds the stack we guessed.

The general idea is that instead of pushing symbols onto the stack, we'll pop the previous guesses, checking that they match up. There are several technical details to this phase.

First, we must encode the top stack symbol for the simulation of P into the state, since we are now using the stack itself to hold guesses. For example, in the beginning of this phase P would be reading Z_0 so we transition from state p to p_{Z_0} to indicate that we are reading Z_0. If we "push" a C by popping the C below the marker, we'll transition to state p_C to record the the top symbol (C is no longer on the stack).

Second, when pushing a symbol in this phase, there are two options. If we guess that the symbol we're pushing will eventually be popped during this phase, we push it on top of the marker and simulate P normally above the marker. If we guess that the symbol we're pushing will be the last one pushed at this position, then we verify the previous guess by popping the symbol below the marker (assuming they match up). Here is an example:

Begining of phase	p_{Z_0}	$\#CBAZ_0$
Arbitrary buildup	p_{Z_0}	$XYZ\#CBAZ_0$
...	p_{Z_0}	...
Eventually return	p_{Z_0}	$\#CBAZ_0$
"Push" a C	p_C	$\#BAZ_0$
...	p_C	...
"Push" a B	p_B	$\#AZ_0$
...	p_B	...
"Push" an A	p_A	$\#Z_0$
Accept!	p_A	

You might notice that there are even more technical details. The example I've shown pops the symbol *below* the marker in one step. We can simulate this with three steps: (1) pop the marker, (2) pop the symbol and (3) replace the marker. Likewise, a transition in δ may push multiple symbols — we can pop multiple symbols by breaking this into several steps using intermediate states.

Theorem: CFLs are not closed under intersection

If L_1 and L_2 are CFLs, then $L_1 \cap L_2$ may not be a CFL.

Proof

1. $L_1 = \{a^n b^n a^m \mid n, m \geq 0\}$ is generated by the following CFG:

   $S \rightarrow XA$

   $X \rightarrow aXb \mid \Lambda$

   $A \rightarrow Aa \mid \Lambda$

2. $L_2 = \{a^n b^m a^m \mid n, m \geq 0\}$ is generated by the following CFG:

   $S \rightarrow AX$

   $X \rightarrow aXb \mid \Lambda$

   $A \rightarrow Aa \mid \Lambda$

3. $L_1 \cap L_2 = \{a^n b^n a^n \mid n \geq 0\}$, which is known not to be a CFL (pumping lemma).

Theorem: CFLs are not closed under complement

If L_1 is a CFL, then $\overline{L_1}$ may not be a CFL.

Proof

They are closed under union. If they are closed under complement, then they are closed under intersection, which is false.

More formally,

1. Assume the complement of every CFL is a CFL.
2. Let L_1 and L_2 be 2 CFLs.
3. Since CFLs are close under union, and we are assuming they are closed under complement,
$$\overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2$$
is a CFL.
4. However, we know there are CFLs whose intersection is not a CFL.
5. Therefore, our assumption that CFLs are closed under complement is false.

Example

This does *not mean* that the complement of a CFL is *never* a CFL.

- Let $L_1 = \{a^n b^n a^n | n \geq 0\}$, which is not a CFL.
- $\overline{L_1}$ is a CFL.
- We show this by constructing it as the union of 5 CFLs.
  - $Mpq = (a^+)(a^n b^n)(a^+) = \{a^p b^q a^r \mid p > q\}$
  - $Mqp = (a^n b^n)(b^+)(a^+) = \{a^p b^q a^r \mid p < q\}$
  - $Mqr = (a^+)(b^+)(b^n a^n) = \{a^p b^q a^r \mid q > r\}$
  - $Mqr = (a^+)(b^n a^n)(a^+) = \{a^p b^q a^r \mid q < r\}$
  - $M = \overline{a^+ b^+ a^+}$ = all words not of the form $a^p b^q a^r$.

  Let $L = M \cup Mpq \cup Mqp \cup Mqr \cup Mqr$.
- Since $M \subseteq L$, $\overline{L}$ contains only words of the form $a^p b^q a^r$.
- $\overline{L}$ cannot contain words of the form $a^p b^q a^r$, where $p < q$.
- $\overline{L}$ cannot contain words of the form $a^p b^q a^r$, where $p > q$.
- Therefore $\overline{L}$ only contains words of the form $a^p b^q a^r$, where $p = q$.
- $\overline{L}$ cannot contain words of the form $a^p b^q a^r$, where $q < r$.
- $\overline{L}$ cannot contain words of the form $a^p b^q a^r$, where $q > r$.
- Therefore $\overline{L}$ only contains words of the form $a^p b^q a^r$, where $q = r$.
- Since $p = q$ and $q = r$, $\overline{L}$ contains words of the form $a^n b^n a^n$, which is not context-free.

Set Difference

Proposition 6. *If L_1 is a CFL and L_2 is a CFL then $L_1 \setminus L_2$ is not necessarily a CFL*

Proof. Because CFLs not closed under complementation, and complementation is a special case of set difference. *(How?)* □

Proposition 7. *If L is a CFL and R is a regular language then $L \setminus R$ is a CFL*

Proof. $L \setminus R = L \cap \overline{R}$ □

Theorem 3.1 (3.5.4) *Context-free languages are not closed under intersection or complement.*

Proof: Let $L_1 = \{a^m b^m c^n : m, n \geq 0\}$ and let $L_2 = \{a^m b^n c^n : m, n \geq 0\}$.

- Then both L_1 and L_2 are context-free.
- However, their intersection $L_1 \cap L_2$ is $\{a^n b^n c^n : n \geq 0\}$ which is not context-free.

For complement, note that $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ where $\overline{L}$ is the complement of L.

- If context-free languages were closed under complement, they would also be closed under intersection.
- Therefore context-free languages are not closed under complementation because they are not closed under intersection.

2. Show that half is not closed for CFL's.

Let $L = \{ a^n b^n c^i d d^{3i} d \mid n>0, i>0 \}$

Since CFL is closed under intersection with regular languages, we can do

$half(L) \cap a^*b^*c^*d$

which splits L into two halves:

$a^n b^n c^i d$ and $d^{3i} d$

In order for these to be of equal length, i = n,

So $half(L) \cap a^*b^*c^*d = a^n b^n c^n d$, which we know is not a CFL.

Therefore half(L) is not closed for CFLs.

Note: There are many choices for L which will work.

Decision Properties of CFL

Testing Emptiness

Testing Membership

Informatika

Testing Emptiness

- **We already did this.**
- **We learned to eliminate variables that generate no terminal string.**
- **If the start symbol is one of these, then the CFL is empty; otherwise not.**

Testing Membership

- **Want to know if string w is in L(G).**
- **Assume G is in CNF.**
  - Or convert the given grammar to CNF.
  - w = ϵ is a special case, solved by testing if the start symbol is nullable.
- **Algorithm (*CYK*) is a good example of dynamic programming and runs in time $O(n^3)$, where n = |w|.**

Testing Membership

- **Want to know if string w is in L(G).**
- **Assume G is in CNF.**
  - Or convert the given grammar to CNF.
  - w = ϵ is a special case, solved by testing if the start symbol is nullable.
- **Algorithm (*CYK*) is a good example of dynamic programming and runs in time $O(n^3)$, where n = |w|.**

CYK Algorithm

- **Let w = $a_1 \ldots a_n$.**
- **We construct an n-by-n triangular array of sets of variables.**
- **X_{ij} = {variables A | A =>* $a_i \ldots a_j$}.**
- **Induction on j–i+1.**
  - The length of the derived string.
- **Finally, ask if S is in X_{1n}.**

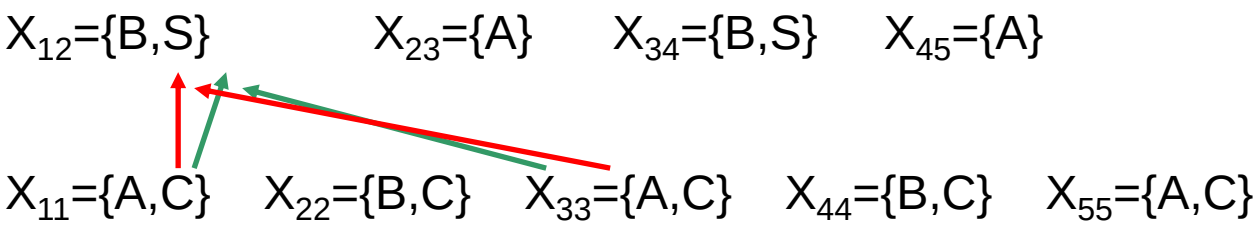
CYK Algorithm – (2)

- **Basis**: $X_{ii} = \{A \mid A \rightarrow a_i \text{ is a production}\}$.
- **Induction**: $X_{ij} = \{A \mid$ there is a production $A \rightarrow BC$ and an integer k, with $i \leq k < j$, such that B is in X_{ik} and C is in $X_{k+1,j}\}$.

Example: CYK Algorithm

Grammar: S -> AB, A -> BC | a, B -> AC | b, C -> a | b

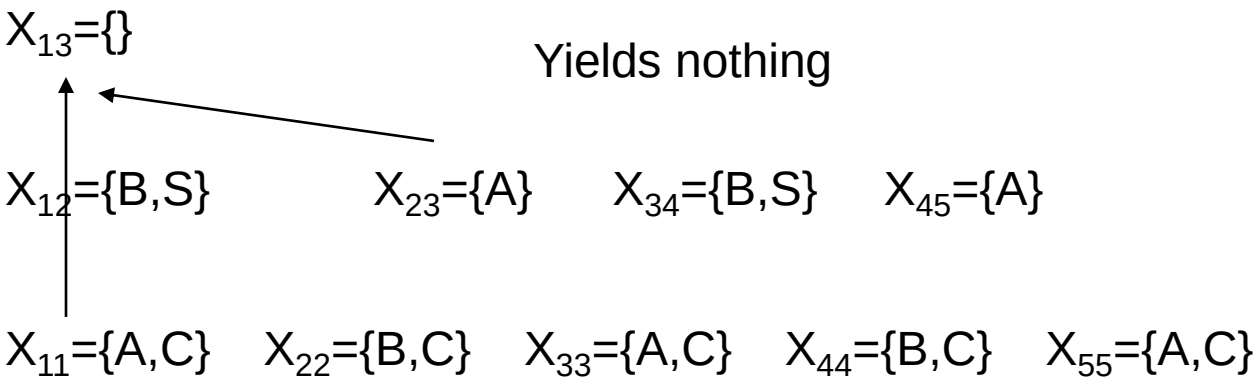
String w = ababa



Example: CYK Algorithm

Grammar: S -> AB, A -> BC | a, B -> AC | b, C -> a | b

String w = ababa



Example: CYK Algorithm

Grammar: S -> AB, A -> BC | a, B -> AC | b, C -> a | b

String w = ababa

X_{13}={A} X_{24}={B,S} X_{35}={A}

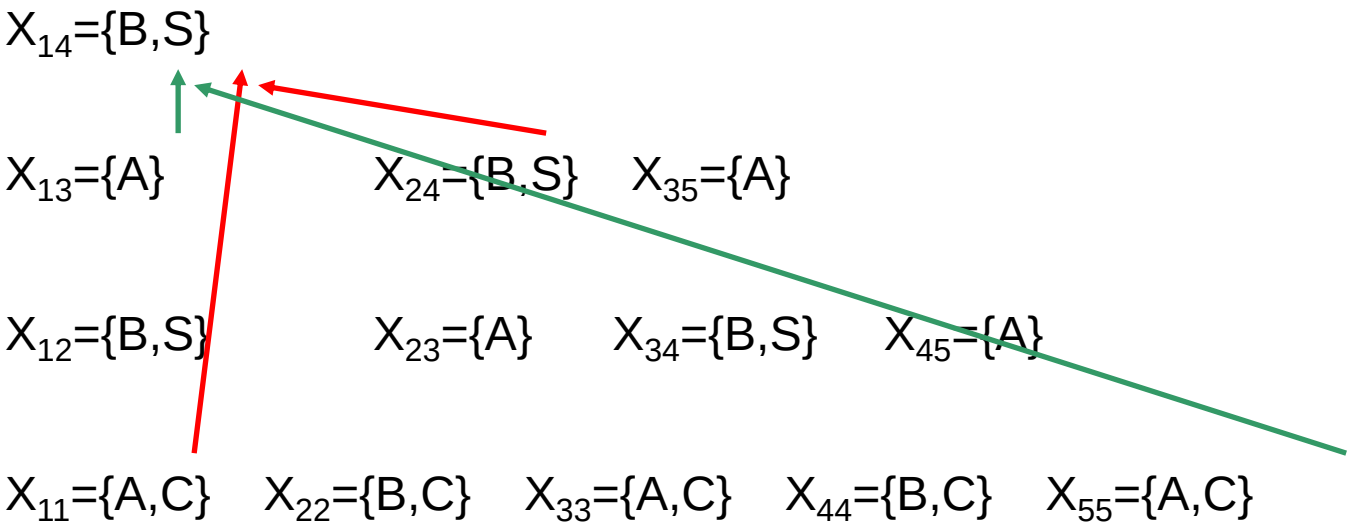
X_{12}={B,S} X_{23}={A} X_{34}={B,S} X_{45}={A}

X_{11}={A,C} X_{22}={B,C} X_{33}={A,C} X_{44}={B,C} X_{55}={A,C}

Example: CYK Algorithm

Grammar: S -> AB, A -> BC | a, B -> AC | b, C -> a | b

String w = ababa



Example: CYK Algorithm

Grammar: S -> AB, A -> BC | a, B -> AC | b, C -> a | b

String w = ababa

$X_{15}=\{A\}$

$X_{14}=\{B,S\}$ $X_{25}=\{A\}$

$X_{13}=\{A\}$ $X_{24}=\{B,S\}$ $X_{35}=\{A\}$

$X_{12}=\{B,S\}$ $X_{23}=\{A\}$ $X_{34}=\{B,S\}$ $X_{45}=\{A\}$

$X_{11}=\{A,C\}$ $X_{22}=\{B,C\}$ $X_{33}=\{A,C\}$ $X_{44}=\{B,C\}$ $X_{55}=\{A,C\}$

Top Down

S S

LHS

A B B C

Example

S → AB | BC

A → BA | a

B → CC | b

C → AB | a

Bottom Up

RHS

A C

a a a b a

Cocke-Younger-Kasami Algorithm

- **Bottom Up Parser**
- **Bentuk CFG diubah ke CNF**

Normal Form

Every production is of type

1) X → YZ

2) X → a

3) S → ε

Example

S → AB | BC

A → BA | a

B → CC | b

C → AB | a

Algoritma CYK - Inisialisasi

- **Basis: iterasi = 1**
  - Seluruh non terminal atau variabel yang dapat diperoleh dari setiap aturan produksi

S → AB | BC

A → BA | a

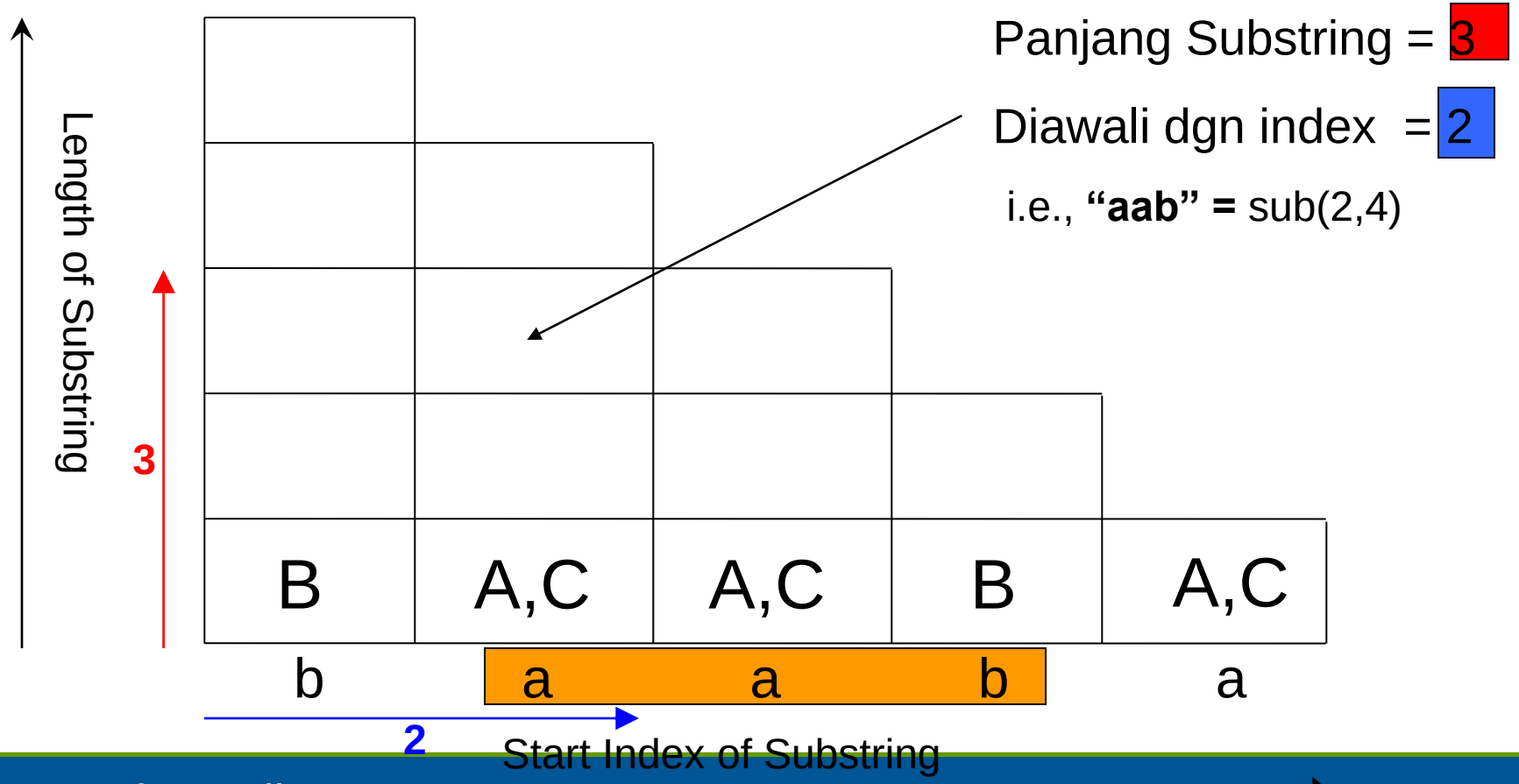
B → CC | b

C → AB | a

B	A,C	A,C	B	A,C
b	a	a	b	a

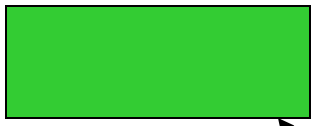
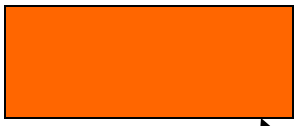
Algoritma CYK– Tabel

- **Setiap sel: Variable dari substring**



CYK – Loop (k>1)

- **Untuk k = 2**
- **Contoh**
  - sub(1,2) = “ba”
  - “ba” = “b” + “a”

=  + 

S → AB | BC

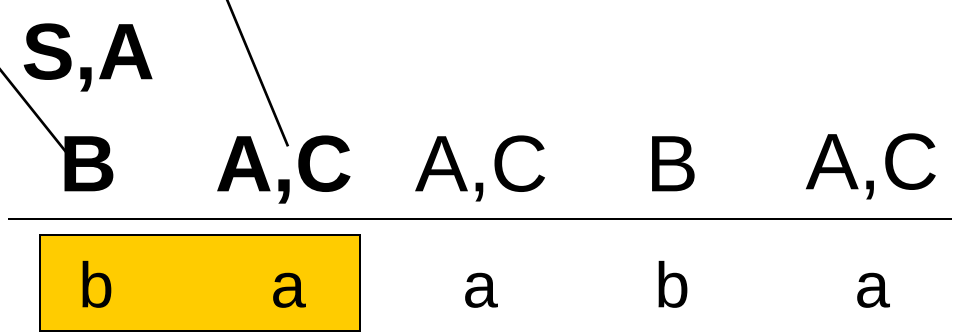
A → BA | a

B → CC | b

C → AB | a

- Yg mungkin:

  BA | BC
- ➔ Variable A,S
  - karena A→BA, S→BC



CYK – Loop (k>1)

- **Utk setiap substring**
  - Dekomposisi ke two substrings
- **Contoh**

  sub(2,4) = “aab”

  = sub(2,2) + sub(3,4)

  = sub(2,3) + sub(4,4)
- Possible:

  AS, AC, CS, CC, BB

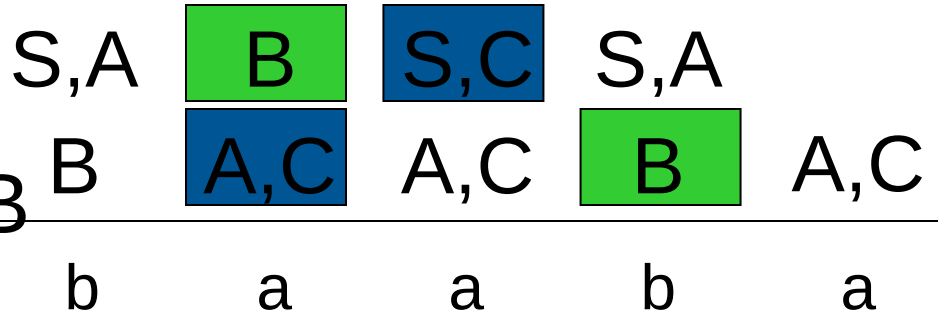
  Maka **B** adalah hasilnya

S → AB | BC

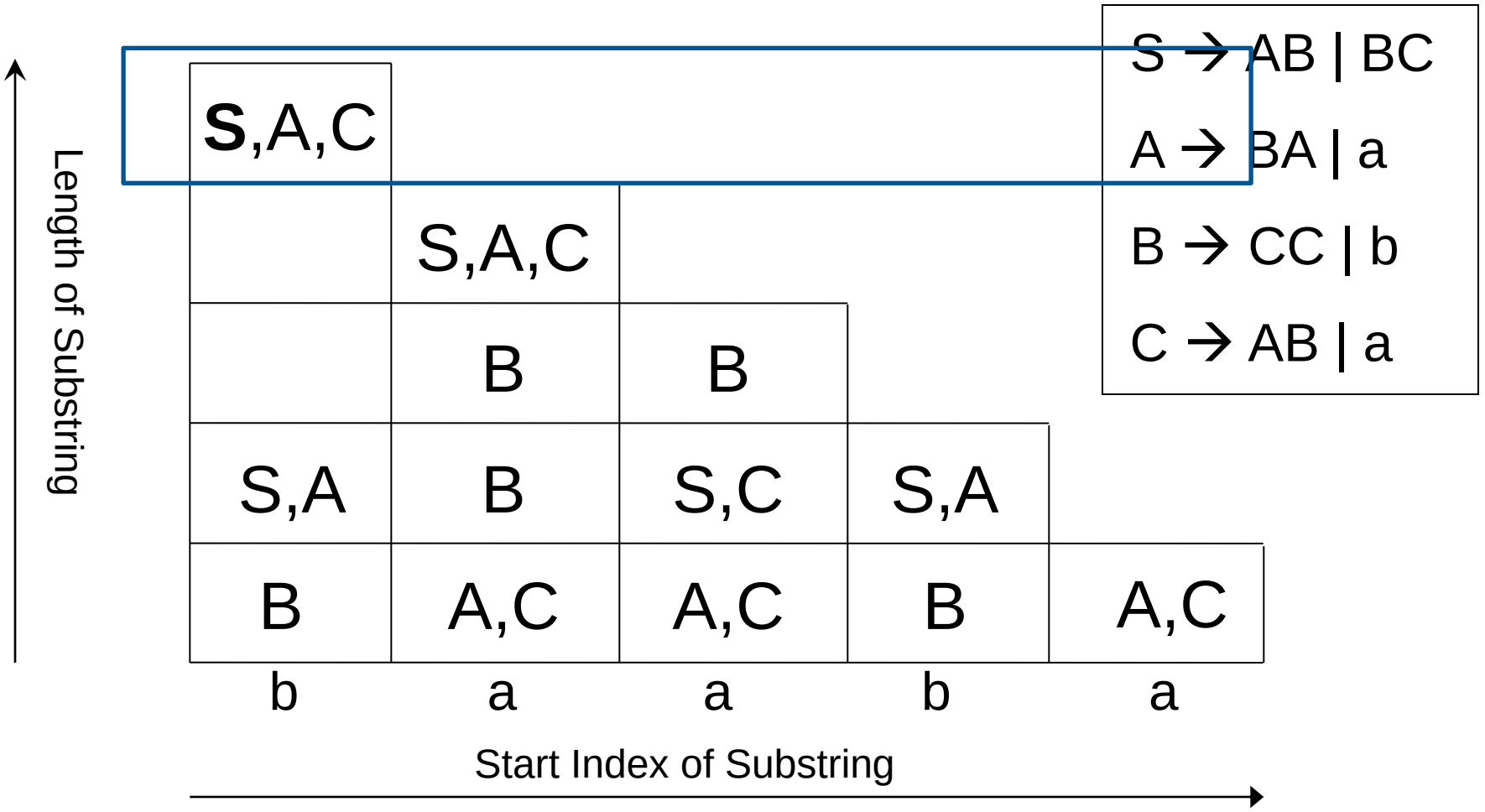
A → BA | a

B → CC | b

C → AB | a

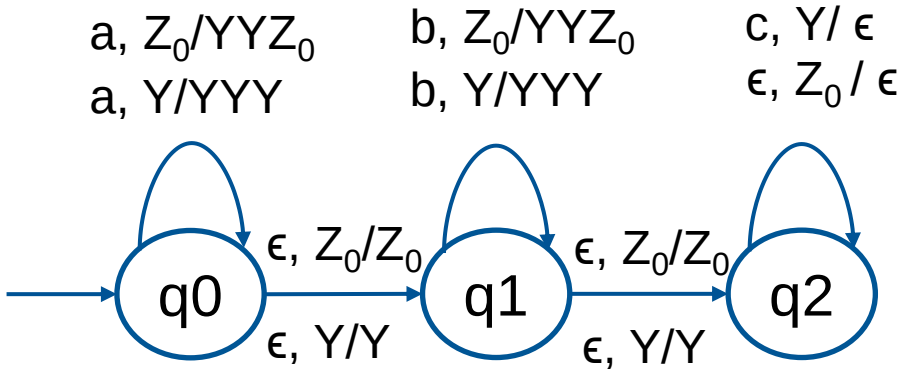


CYK Algorithm – Parse Tree



Buatlah PDA untuk menerima bahasa di bawah ini dengan empty-stack
$L(G) = \{a^n b^m c^{2(n+m)} : n>=0, m>=0\}$
Tuliskan setiap komponen PDA berikut P = $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ dimana transition function dituliskan berupa bentuk δ (state asal, simbol input, top of stack) = {(state tujuan, string of stack symbol)

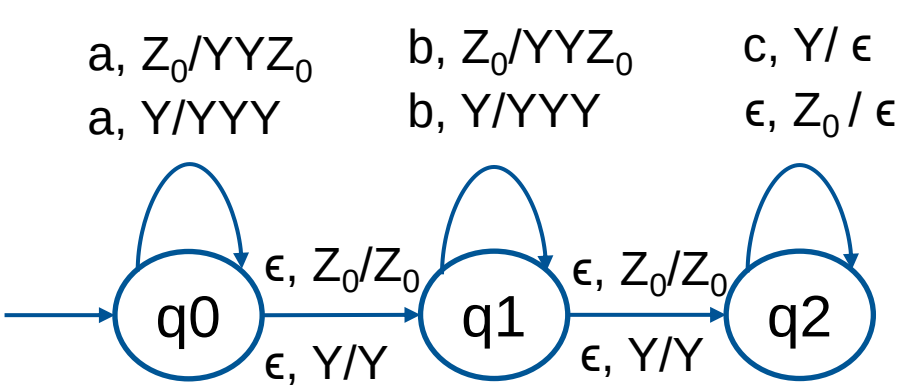
$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b, c\}$
$\Gamma = \{Y, Z_0\}$
$\delta =$
q_0
Z_0



$\delta(q_0, a, Z_0) = \{(q_0, YYZ_0)\}$
$\delta(q_0, a, Y) = \{(q_0, YYY)\}$
$\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$
$\delta(q_0, \epsilon, Y) = \{(q_1, Y)\}$

$\delta(q_1, b, Z_0) = \{(q_1, YYZ_0)\}$
$\delta(q_1, b, Y) = \{(q_1, YYY)\}$
$\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$
$\delta(q_1, \epsilon, Y) = \{(q_2, Y)\}$

$\delta(q_2, c, Y) = \{(q_2, \epsilon)\}$
$\delta(q_2, c, Z_0) = \{(q_2, \epsilon)\}$



CFG

$S \rightarrow [q_0\ Z_0\ q_0]$

$S \rightarrow [q_0\ Z_0\ q_1]$

$S \rightarrow [q_0\ Z_0\ q_2]$

pop

$[q_2\ Y\ q_2] \rightarrow c$

$[q_2\ Z_0\ q_2] \rightarrow \epsilon$

Tetap/ganti

$[q_0\ Z_0\ q_0] \rightarrow [q_1\ Z_0\ q_0]$
$[q_0\ Z_0\ q_1] \rightarrow [q_1\ Z_0\ q_1]$
$[q_0\ Z_0\ q_2] \rightarrow [q_1\ Z_0\ q_2]$
$[q_0\ Y\ q_0] \rightarrow [q_1\ Y\ q_0]$
$[q_0\ Y\ q_1] \rightarrow [q_1\ Y\ q_1]$
$[q_0\ Y\ q_2] \rightarrow [q_1\ Y\ q_2]$
$[q_1\ Z_0\ q_0] \rightarrow [q_2\ Z_0\ q_0]$
$[q_1\ Z_0\ q_1] \rightarrow [q_2\ Z_0\ q_1]$
$[q_1\ Z_0\ q_2] \rightarrow [q_2\ Z_0\ q_2]$
$[q_1\ Y\ q_0] \rightarrow [q_2\ Y\ q_0]$
$[q_1\ Y\ q_1] \rightarrow [q_2\ Y\ q_1]$
$[q_1\ Y\ q_2] \rightarrow [q_2\ Y\ q_2]$

push

$[q_0\ Z_0\ q_0] \rightarrow a\ [q_0\ Y\ q_0]\ [q_0\ Y\ q_0]\ [q_0\ Z_0\ q_0]$
$[q_0\ Z_0\ q_0] \rightarrow a\ [q_0\ Y\ q_0]\ [q_0\ Y\ q_1]\ [q_1\ Z_0\ q_0]$
$[q_0\ Z_0\ q_0] \rightarrow a\ [q_0\ Y\ q_0]\ [q_0\ Y\ q_2]\ [q_2\ Z_0\ q_0]$
$[q_0\ Z_0\ q_0] \rightarrow a\ [q_0\ Y\ q_1]\ [q_1\ Y\ q_0]\ [q_0\ Z_0\ q_0]$
$[q_0\ Z_0\ q_0] \rightarrow a\ [q_0\ Y\ q_2]\ [q_2\ Y\ q_0]\ [q_0\ Z_0\ q_0]$
dst….

- Target adalah $S \rightarrow [q_0\ Z_0\ q_f]$
- Jika transisi di pop: $\delta(q,0,Z) = (p,\epsilon)$ maka pd CFG: $[qZp] \rightarrow 0$
- Jika transisi tetap atau diganti: $\delta(q,0,Z) = (p,Y)$ maka pd CFG: $[qZ\mathbf{r}] \rightarrow 0\ [pY\mathbf{r}]$ dimana r adalah semua state yg ada pd PDA
- Jika transisi di push: $\delta(q,0,Z) = (p,YZ)$ maka $[qZ\mathbf{r}] \rightarrow 0\ [pY\mathbf{s}]\ [\mathbf{s}Z\mathbf{r}]$ dimana r dan s adalah semua state yg ada pd PDA

Diberikan tata bahasa G berikut dengan E sebagai start symbol. Gunakan algoritma CYK untuk memeriksa apakah ekspresi a * (a + a) anggota Bahasa L(G).

$$E \rightarrow E + T \mid T$$

$$T \rightarrow F \mid T * F$$

$$F \rightarrow (E) \mid a$$

Null Production: tidak ada

Unit Production: E → T dan T → F

E → E + T | T * F | (E) | a
T → T * F | (E) | a
F → (E) | a

Useless variable: tidak ada

Mengubah ke bentuk RHS berupa 2 Variabel atau 1 Terminal

F → (E) | a
G → (
H →)
K → EH
F → GK | a

T → T * F
M → *
N → MF
T → TN
T → GK | a

E → E + T
O → +
P → OT
E → EP
E → TN | GK | a

G → (
H →)
K → EH
F → GK | a

M → *
N → MF
T → TN
T → GK | a

O → +
P → OT
E → EP
E → TN | GK | a

Start Variable: **E**

a * (a + a)

7	**E**						
6		N					
5			F				
4				K			
3				E			
2					P	K	
1	T,F,E	M	G	T,F,E	O	T,F,E	H
	a	*	(	a	+	a	)