Context Free Languages

Kuliah Teori Bahasa dan Automata
Program Studi Ilmu Komputer
Fasilkom UI

Prepared by:
Rahmad Mahendra

Revised by:
Suryana Setiawan

Context-Free Grammar

- Bahasa L adalah *context-free* jika dan hanya jika L dapat dibentuk oleh suatu *context-free grammar* (CFG) G.
- Pada CFG, *left-hand side* pada setiap *rule* harus berupa simbol non-terminal tunggal. Sedangkan *right-hand side* bisa berupa urutan simbol apapun (non-terminal maupun terminal, boleh string kosong).
- Contoh rule yang valid pada CFG
  - $S \rightarrow a$ $S \rightarrow bSb$
  - $S \rightarrow T$ $S \rightarrow aaSSbT$
- Contoh rule yang tidak valid pada CFG
  - $aSb \rightarrow aTb$ $a \rightarrow \varepsilon$
  - $ST \rightarrow bb$

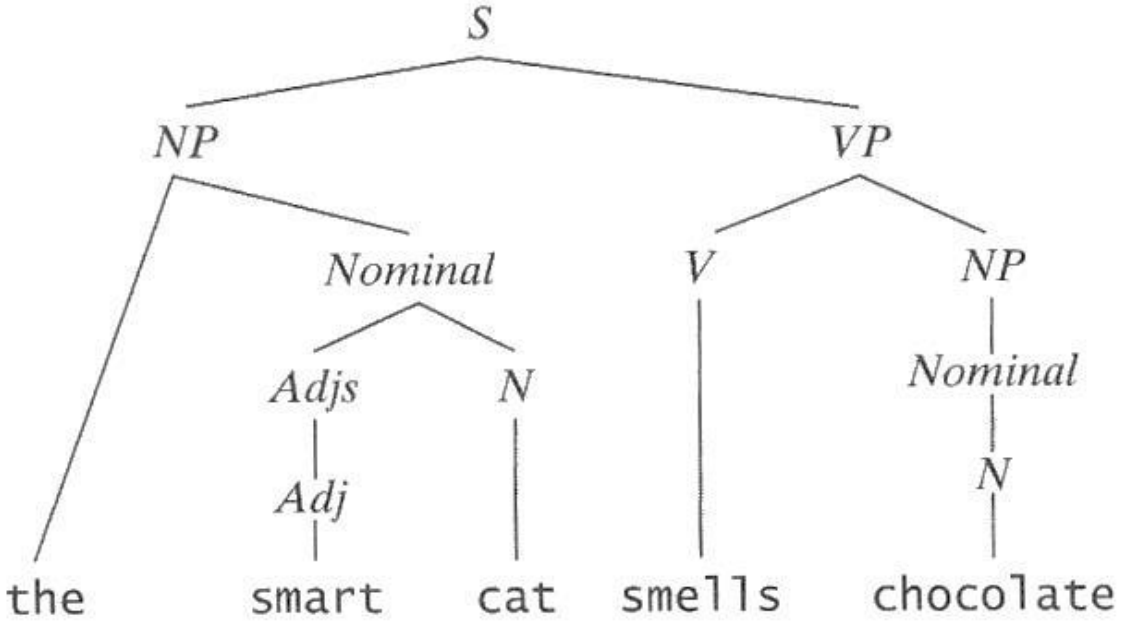
Parse Tree

- Ingat bahwa:
  - $x =>_G y$ adalah relasi *derive in-one-step* string x menjadi y (dimana $x, y \in V^*$) melalui aplikasi suatu rule dalam R_G.
  - Deretan relasi $S =>_G x_1 =>_G x_2 \ldots => w$ (atau disingkat $S =>_G^* w$) disebut juga derivasi S menjadi w
- Untuk menunjukkan struktur dan proses terbentuknya string w menurut rule-rule dalam G tsb, derivasi ditunjukkan sebagai tree
  - Disebut *derivation tree* atau *parse tree*
- **Parser**: algoritma untuk mendapatkan *parse tree* dari suatu string menurut suatu grammar.

Definisi Parse Tree

- Suatu parse tree dalam derivasi menurut grammar $G = (V, \Sigma, R, S)$, adalah *rooted, ordered tree* yang mana:
  - Setiap *leaf node* berlabelkan suatu elemen $(\Sigma \cup \{\varepsilon\})$,
  - *Root node* berlabel S,
  - Setiap node yang lain berlabel elemen-elemen $(V - \Sigma)$, dan
  - Jika m adalah *nonleaf node* berlabel X, dan anak-anak m berlabel $x_1, x_2, \ldots, x_n$, maka R berisi rule $X \rightarrow x_1, x_2, \ldots, x_n$.

$S \rightarrow NP\ VP$
$NP \rightarrow$ the $Nominal$ | a $Nominal$ | $Nominal$ | $ProperNoun$ | $NP\ PP$
$Nominal \rightarrow N$ | $Adjs\ N$
$N \rightarrow$ cat | dogs | bear | girl | chocolate | rifle
$ProperNoun \rightarrow$ Chris | Fluffy
$Adjs \rightarrow Adj\ Adjs$ | Adj
$Adj \rightarrow$ young | older | smart
$VP \rightarrow V$ | $V\ NP$ | $VP\ PP$
$V \rightarrow$ like | likes | thinks | shot | smells
$PP \rightarrow Prep\ NP$
$Prep \rightarrow$ with



Terminologi

- Branching Factor (BF)
  - dari grammar *G*: banyaknya simbol dalam ruas kanan terpanjang dari rule-rule dalam R_G.
  - dari *parse tree*: banyaknya simbol dalam ruas kanan terpanjang dari rule-rule yang digunakan dl *parse tree*.
  - BF dari *parse tree* (dari *G*) ≤ BF dari grammar *G*.
- Generative Capacity
  - **Weak generative capacity** dari *G*: himpunan string (*L*(*G*)) yang dihasilkan *G*.
  - **Strong generative capacity** dari *G*: himpunan *parse tree* yang dihasilkan *G*.

Left vs. Right derivation

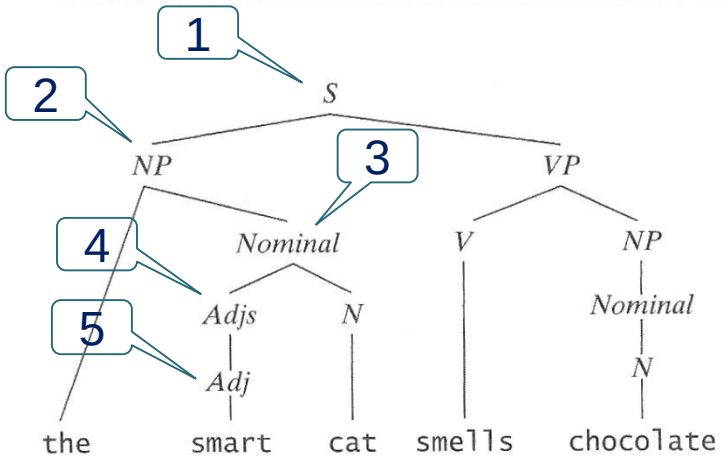
- *Parse tree* tidak menunjukkan urutan penerapan rule-rule yang digunakan.
  Untuk acuan selanjutnya terdapat dua kaidah dalam derivasi:
  - **Left-most derivation**: derivasi berikutnya diterapkan pada nonterminal paling kiri hingga paling kanan.
  - **Right-most derivation**: derivasi berikutnya diterapkan pada nonterminal paling kanan hingga paling kiri.

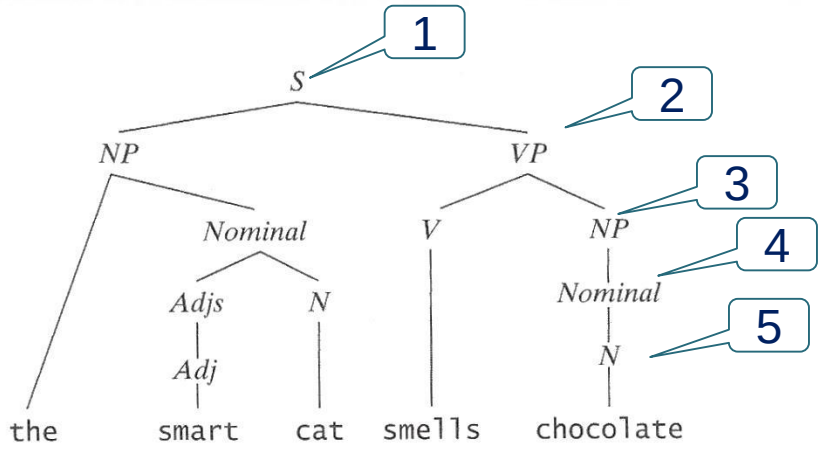
A left-most derivation is:

$S \Rightarrow NP\ VP \Rightarrow$ The $Nominal\ VP \Rightarrow$ The $Adjs\ N\ VP \Rightarrow$ The $Adj\ N\ VP \Rightarrow$
The smart $N\ VP \Rightarrow$ the smart cat $VP \Rightarrow$ the smart cat $V\ NP \Rightarrow$
the smart cat smells $NP \Rightarrow$ the smart cat smells $Nominal \Rightarrow$
the smart cat smells $N \Rightarrow$ the smart cat smells chocolate

A right-most derivation is:

$S \Rightarrow NP\ VP \Rightarrow NP\ V\ NP \Rightarrow NP\ V\ Nominal \Rightarrow NP\ V\ N \Rightarrow NP\ V$ chocolate $\Rightarrow$
NP smells chocolate $\Rightarrow$ the $Nominal$ smells chocolate $\Rightarrow$
the $Adjs\ N$ smells chocolate $\Rightarrow$ The $Adjs$ cat smells chocolate $\Rightarrow$
the Adj cat smells chocolate $\Rightarrow$ the smart cat smells chocolate





Unreachable/Nonproductive Rule

- Perhatikan grammar berikut

  $G = (\{S, A, B, C, D, \text{a}, \text{b}\}, \{\text{a}, \text{b}\}, R, S)$ di mana

  $R = \{$

  $S \rightarrow AB \mid AC$ $\quad\quad$ $C \rightarrow \text{b}C\text{a}$

  $A \rightarrow \text{a}Ab \mid \varepsilon$ $\quad\quad$ $D \rightarrow AB$

  $B \rightarrow \text{b}A$

  $\}$

- Pada grammar di atas terdapat dua variabel yang tidak berguna. Simbol non-terminal *C* tidak dapat membentuk string apapun di Σ^*, sedangkan simbol non-terminal *D* tidak dapat dijangkau dengan derivasi apapun dari *S*.
- Simbol *C* disebut **tidak produktif**, sedangkan simbol *D* ***unreachable***
- Simplikasi grammar dapat dilakukan untuk membuang *rule-rule* yang tidak produtif atau yang *unreachable*

Simplikasi CFG: Membuang Nonproductive Rules

Algoritma membuang rule yang tidak produktif

1. Tandai setiap simbol non-terminal pada G sebagai tidak produktif
2. Tandai setiap simbol terminal sebagai produktif
3. Untuk setiap rule berbentuk $X \rightarrow \alpha$
   - Jika α produktif, tandai X sebagai produktif
   - Lakukan proses ini secara iteratif
4. Buang seluruh rule yang memuat simbol-simbol yang tidak produktif, baik di *left-hand side* maupun *right-hand side*
5. Buang simbol-simbol tidak produktif

Simplikasi CFG: Membuang unreachable Rules

Algoritma membuang rule yang *unreachable*

1. Tandai *S* pada *G* sebagai *reachable*
2. Tandai setiap simbol non-terminal sebagai *unreachable*
3. Untuk setiap rule berbentuk $X \rightarrow \alpha A \beta$

   (di mana $A \in V - \Sigma$ dan $\alpha, \beta \in V^*$)

   - Jika *X reachable*, tandai *A* sebagai *reachable*
   - Lakukan proses ini secara iteratif
4. Buang seluruh rule yang memuat simbol-simbol yang *unreachable*, di *left-hand side* maupun *right-hand side*
5. Buang simbol-simbol yang *unreachable*

Normal Form

- Bentuk normal (*normal form*) berguna untuk memudahkan pekerjaan.
  - Contoh: normal form untuk query memudahkan pengolahan basis data, normal form untuk formula logika memudahkan *reasoning* dalam sistem AI.
- Misalkan *C* adalah sekumpulan objek data. *F* disebut sebagai *normal form* dari *C* jika dan hanya jika:
  1. Untuk setiap elemen *c* dalam *C*, terdapat beberapa elemen *f* dalam *F*, sehingga *f* ekuivalen dengan *c* dalam mengerjakan suatu *task*.
  2. *F* lebih sederhana dari *C*. Sejumlah *task* lebih mudah dikerjakan oleh *F* dibandingkan dengan *C*.

Chomsky Normal Form

- Salah satu bentuk normal yang sering dipakai untuk CFG adalah *Chomsky Normal Form* (CNF).
- Suatu grammar $G = (V, \Sigma, R, S)$ dalam CNF apabila semua rule R berbentuk
  - $X \rightarrow a$, di mana $a \in \Sigma$, atau
  - $X \rightarrow BC$, di mana $B, C \in V - \Sigma$
- Semua parse tree yang dibangkitkan oleh grammar dalam bentuk CNF memiliki *branching factor* 1 atau 2. Mengapa?
- Setiap derivasi string w mengandung $|w| - 1$ pengaplikasian rule berbentuk $X \rightarrow BC$ dan $|w|$ pengaplikasian rule berbentuk $X \rightarrow a$

Chomsky Normal Form

- Teorema

  Diberikan suatu context-free grammar G, terdapat suatu bentuk Chomsky normal form G_C, sehingga $L(G_C) = L(G) - \{\varepsilon\}$.

- Pembuktian:

  Terdapat algoritma konversi suatu grammar G ke dalam Chomsky normal form.

Konversi CFG dalam CNF

Diberikan $G = (V, \Sigma, R, S)$

- G_C diperoleh dari hasil eliminasi ε-rule pada G.
- G_C diperoleh dari hasil eliminasi rule-rule pada G_C yang mana RHSnya mengandung sebuah simbol non-terminal (*unit production*).
- G_C diperoleh dari hasil eliminasi rule-rule pada G_C yang mana RHSnya memiliki panjang > 1 dan mengandung simbol terminal (*mixed production*).
- G_C diperoleh dari hasil eliminasi rule-rule pada G_C yang mana RHSnya memiliki panjang > 2 (long RHS).
- $L(G_C) = L(G) - \{\varepsilon\}$

I. Eliminasi ε-Rule

- Beberapa definisi untuk Eliminasi ε-Rule
  - **Nullable Variable**:
    Suatu variable X adalah *nullable* **iff either**
    (1) tdp. $X \rightarrow \varepsilon$, atau
    (2) tdp. $X \rightarrow PQR\ldots$ dimana setiap dari
    $P, Q, R, \ldots$ adalah *nullable*
  - **Modifiable rule**:
    suatu rule *modifiable* **iff**
    rule berbentuk $P \rightarrow \alpha Q \beta$, dengan
    Q *nullable* dan $\alpha, \beta \in V^*$

Algoritma Eliminasi ε-Rule

- Jika $G = (V, \Sigma, R, S)$ adalah suatu CFG, grammar G' dapat dibentuk tanpa berisi ε-rule sehingga $L(G') = L(G) - \{\varepsilon\}$, sbb:
  1. $G' = G$
  2. $N = \{\}$
  3. Dalam iterasi:
     Temukan *nullable variable* berikutnya dalam G' dan tambahkan dalam N hingga N tidak berubah (N himp semua *nullable variable*)
  4. Dalam iterasi:
     Untuk tiap *modifiable rule* $P \rightarrow \alpha Q \beta$, dengan $\alpha\beta \neq \varepsilon$ dan $\alpha\beta \neq P$ (jelas $Q \in N$),
     tambahkan dalam G' rule baru $P \rightarrow \alpha\beta$
  5. Hapus dari G' setiap rule berbentuk $X \rightarrow \varepsilon$

Contoh 1 (Eliminasi ε-Rule)

- Untuk grammar G dengan rule-rule:

  $S \rightarrow aTa$
  $T \rightarrow ABC$
  $A \rightarrow aB \mid C$
  $B \rightarrow Bb \mid C$
  $C \rightarrow c \mid \varepsilon$

- Sesuai urutan diperolehnya, $N = \{C, A, B, T\}$
- Modifier rules: setiap rule, kecuali $C \rightarrow c$
- G' = G ditambah rule-rule

  $S \rightarrow aa$
  $T \rightarrow BC \mid AC \mid AB \mid A \mid B \mid C$
  $A \rightarrow a$
  $B \rightarrow b$

  dan dengan menghapus $C \rightarrow \varepsilon$

Mengembalikan ε

- Seringkali ε memang perlu untuk dapat dihasilkan.
- Jika $G = (V, \Sigma, R, S_G)$ adalah suatu CFG, grammar G'' dapat dibentuk sehingga $L(G'') = L(G)$, sbb:
  - G'' diperoleh dari eliminasi ε-Rule sebelumnya.
  - Jika S_G (start symbol dalam G) adalah *nullable variable*, maka, tambahkan nonterminal baru S'' sebagai start symbol yang baru dalam G"
  - Tambahkan $S'' \rightarrow S_G \mid \varepsilon$

Contoh 2 (Eliminasi ε-Rule)

- Untuk Bal

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

- Eliminasi ε-Rule menghasilkan Bal'

  $S \rightarrow (S)$
  $S \rightarrow ()$
  $S \rightarrow SS$

- Setelah mengembalikan ε

  $S'' \rightarrow \varepsilon$
  $S'' \rightarrow S$
  $S \rightarrow (S)$
  $S \rightarrow ()$
  $S \rightarrow SS$

II. Eliminasi Unit Productions

- Jika $G = (V, \Sigma, R, S)$ adalah suatu CFG, grammar G' dapat dibentuk tanpa berisi *unit productions*, sbb:
  1. G' = G
  2. Dalam iterasi sampai tidak terdapat *unit productions* dalam G':
     a. Pilih *unit production* $X \rightarrow Y$
     b. Hapus *rule* tersebut dari G'
     c. Pertimbangkan *rule* yang masih dalam G'

        Untuk setiap rule $Y \rightarrow \beta$ di mana $\beta \in V^*$:

        Tambahkan rule $X \rightarrow \beta$ ke dalam G' kecuali rule tersebut pernah dihapus sebelumnya

     Urutan penghapusan *unit productions* tidak masalah.

Contoh (Eliminasi Unit Productions)

Grammar G dengan rule-rule:

$S \rightarrow XY$ $\quad\quad$ $A \rightarrow B \mid a$

$X \rightarrow A$ $\quad\quad$ $B \rightarrow b$

$Y \rightarrow T$ $\quad\quad$ $T \rightarrow Y \mid c$

- Hapus rule $X \rightarrow A$

  Karena $A \rightarrow B \mid$ a, tambahkan rule $X \rightarrow B \mid$ a
- Hapus rule $X \rightarrow B$

  Tambahkan rule $X \rightarrow$ b
- Hapus rule $Y \rightarrow T$

  Tambahkan rule $Y \rightarrow Y \mid$ c
- Hapus rule $Y \rightarrow Y$

  Pertimbangkan untuk menambah rule $Y \rightarrow T$, tetapi tidak jadi karena rule $Y \rightarrow T$ pernah dihapus sebelumnya

Contoh (Eliminasi Unit Productions)

Grammar G dengan rule-rule:

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

- Hapus rule $A \rightarrow B$

  Tambahkan rule $A \rightarrow b$
- Hapus rule $T \rightarrow Y$

  Tambahkan rule $T \rightarrow c$ (rule sudah ada di G')

Diperoleh grammar G' tanpa unit productions dengan rule-rule:

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

III. Eliminasi Mixed Productions

- Jika $G = (V, \Sigma, R, S)$ adalah suatu CFG, grammar G' dapat dibentuk tanpa berisi *mixed productions*, sbb:
  1. G' = G
  2. Buatlah simbol non-terminal baru untuk setiap simbol terminal dalam Σ (misalnya T_a untuk a)
  3. Modifikasi setiap rule dalam G' yang memiliki RHS panjang > 1 dan mengandung simbol terminal dengan cara substitusi a dengan T_a
  4. Tambahkan rule $T_a \rightarrow a$ ke dalam G'

Contoh (Eliminasi Mixed Production)

- Grammar G dengan rule-rule:

$$A \rightarrow \text{a}$$
$$A \rightarrow \text{a}B$$
$$A \rightarrow B\text{a}C$$
$$A \rightarrow B\text{b}C$$

- Grammar G' tanpa mixed productions, dengan rule-rule:

$$A \rightarrow \text{a}$$
$$A \rightarrow T_{\text{a}}B$$
$$A \rightarrow BT_{\text{a}}C$$
$$A \rightarrow BT_{\text{b}}C$$
$$T_{\text{a}} \rightarrow \text{a}$$
$$T_{\text{b}} \rightarrow \text{b}$$

IV. Eliminasi Long RHS

- Jika $G = (V, \Sigma, R, S)$ adalah suatu CFG, grammar G' dapat dibentuk tanpa mengandung *long* RHS, sbb:
  1. G' = G
  2. Untuk setiap rule r^k berbentuk $A \rightarrow N_1 N_2 N_3 \ldots N_n$ $n > 2$, buat sejumlah $n - 2$ simbol non terminal baru M^k_2 , $M^k_3 \ldots$, dan M^k_{n-1}
  3. Dalam G', substitusi rule r^k menjadi $A \rightarrow N_1 M^k_2$
  4. Ke dalam G', tambahkan rule

$$M^k_2 \rightarrow N_2 M^k_3 \text{ , } M^k_3 \rightarrow N_3 M^k_4 \text{ , } \ldots \text{ , dan } M^k_{n-1} \rightarrow N_{n-1} M^k_n$$

Contoh (Eliminasi Long RHS)

- Grammar G dengan rule-rule:

  $A \rightarrow BCDE$

  $D \rightarrow FG$

- Grammar G' tanpa *long* RHS, dengan rule-rule:

  $A \rightarrow BM_1$

  $M_1 \rightarrow CM_2$

  $M_2 \rightarrow DE$

  $D \rightarrow FG$

Latihan

- Grammar $G = (\{S, A, B, C, \text{a}, \text{c}\}, \{A, B, C\}, R, S)$

  di mana $R = \{S \rightarrow \text{a}AC\text{a}$

  $A \rightarrow B \mid \text{a}$

  $B \rightarrow C \mid \text{c}$

  $C \rightarrow \text{c}C \mid \varepsilon\}$

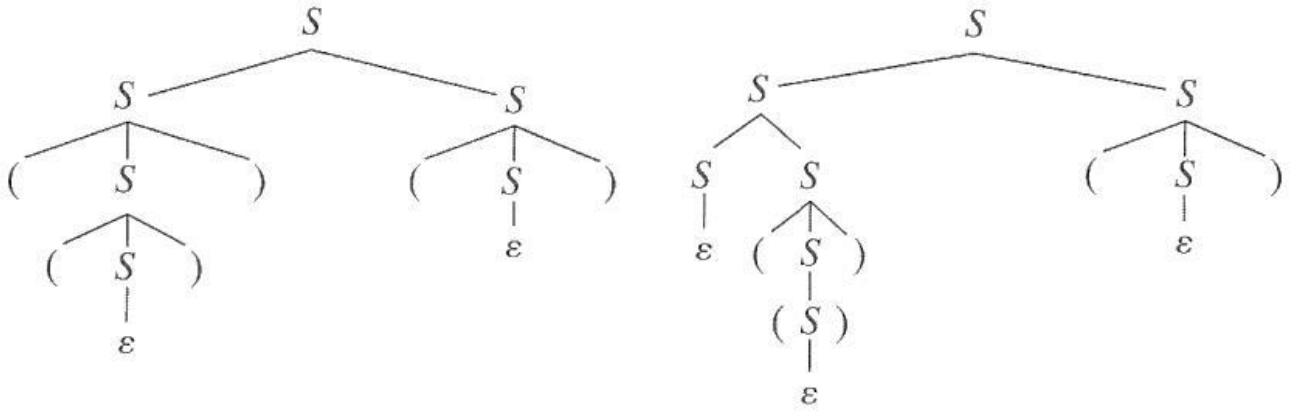
- Ubahlah grammar G ke dalam bentuk normal Chomsky

Ambiguitas

- **Ambiguous grammar**: untuk suatu string dalam $L(G)$, grammar G dapat menghasilkan > 1 parse tree berbeda.
- Contoh: Untuk **Bal** (*balance parantheses*), digunakan rule-rule sbb:

  $S \rightarrow (S) \mid SS \mid \varepsilon$

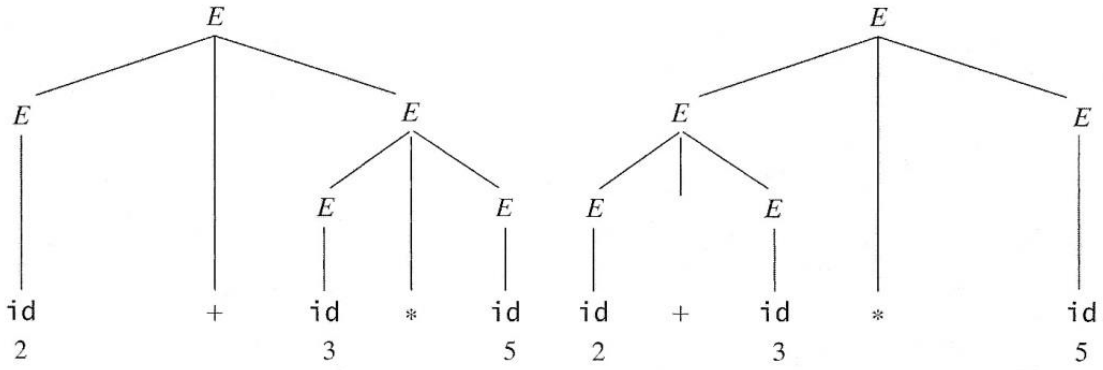
  untuk string (())() terdapat bbrp *parse tree*, diantaranya:



Masalahkah Ambiguitas itu?

- Ekspresi reguler sering pula memiliki ambiguitas tetapi umumnya tidak menjadi masalah karena untuk bahasa reguler struktur derivasi string tidak terlalu penting
- Dalam bahasa Context-free jadi masalah karena seringkali arti dari string terkait stuktur derivasi tsb.
- Contoh: `id+id*id` (dalam mengartikan 2+3*5) dari grammar berikut: “apakah artinya 17 atau 25?”

$$R = \{E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow (E)$$
$$E \rightarrow \texttt{id}\ \}.$$



Inherently Ambiguous

- Terkadang, ambiguitas suatu grammar bisa dihilangkan atau dikurangi, tapi tidak selalu bisa!
- Terdapat CFL yang memang tidak memiliki *unambiguous grammar*; disebut bahasa-bahasa *inherently ambiguous*.
- Contoh:
  $L = \{a^i b^j c^k : i, j, k \geq 0, i = j \text{ atau } j = k\}$
  $= \{a^n b^n c^m : n, m \geq 0\} \cup \{a^n b^m c^m : n, m \geq 0\}$
- Grammar bahasa ini adalah:
  $S \rightarrow S_1 \mid S_2 \quad S_1 \rightarrow S_1 c \mid A \quad S_2 \rightarrow a S_2 \mid B$
  $A \rightarrow aAb \mid \varepsilon \quad B \rightarrow bBc \mid \varepsilon$
- Untuk setiap string dari $\{a^n b^n c^n : n \geq 0\}$ tdp dua derivasi berbeda

Mengurangi Ambiguitas

- Tiga struktur/bentuk rule dari grammar yang sering mengarah pada ambiguitas:
  - Rule dengan ruas kanan ε (atau **ε-Rule**)
    Misalnya: $S \rightarrow \varepsilon$
  - Rule ruas kanan simetris dan memiliki dua copy dari nonterminal di ruas kiri (*symmetric recursive*)
    Misalnya: $S \rightarrow SS$ atau $E \rightarrow E + E$
  - Sejumlah rule yang menghasilkan suffiks/penambahan ambiguous (*ambiguous attachment*)

Ambiguitas: Eliminasi *ε-Rule*

- Sudah dibahas di Normal Forms
- Menghilangkan ε-Rule mengurangi kemungkinan ambiguitas

Ambiguitas: Eliminasi *Symmetric Recursive Rule*

- Menggantikan $S \rightarrow SS$ dengan
  $S \rightarrow SS_1$ (memaksa branching ke kiri) atau
  $S \rightarrow S_1S$ (memaksa branching ke kanan)
- Kemudian modifikasi rule lain yang terkait.
- Untuk contoh Bal, kita gunakan $S \rightarrow SS_1$ lalu modifikasi dengan:
  - menambahkan $S \rightarrow S_1$
  - mengganti $S \rightarrow (S)$ dengan $S_1 \rightarrow (S)$
- mengganti $S \rightarrow ()$ dengan $S_1 \rightarrow ()$
  Menjadi:
  $S'' \rightarrow \varepsilon \mid S$
  $S \rightarrow SS_1 \mid S_1$
  $S_1 \rightarrow (S) \mid ()$

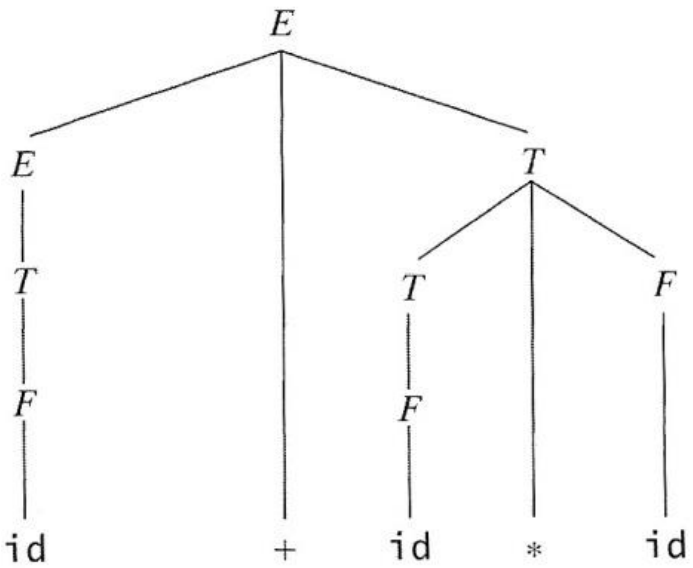
Contoh 3: Ekspresi Aritmetika

- Semula ambiguous:

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

- Setelah menjadi unambiguous

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



Ambiguitas: Menghindari *Ambiguous Attachment*

- Secara umum *ambiguous attachment* terjadi karena adanya rule: $A \rightarrow \alpha$ dan $A \rightarrow \alpha\beta$
- Adanya β menjadikan situasi ambiguous pada $\alpha\alpha\beta$: apakah β terkait α pertama atau α kedua.
- Contoh: "the dangling *else* problem" pada bahasa-bahasa pemrograman,

  <stmt> ::= **if** <cond> **then** <stmt>

  <stmt> ::= **if** <cond> **then** <stmt> **else** <stmt>
- Untuk statemen :
  "**if** cond1 **then if** cond2 *attachment* **then** st1 **else** st2",
  "**else** st2" bagian dari "**if** cond1"atau "**if** cond2"?