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).
  - Apa hubungan antara CFG dengan regular grammar?
- Contoh rule yang valid pada CFG
  - $S \rightarrow \mathrm{a}$ $\quad$ $S \rightarrow \mathrm{b}S\mathrm{b}$
  - $S \rightarrow T$ $\quad$ $S \rightarrow \mathrm{aa}SS\mathrm{b}T$
- Contoh rule yang tidak valid pada CFG
  - $\mathrm{a}S\mathrm{b} \rightarrow \mathrm{a}T\mathrm{b}$ $\quad$ $\mathrm{a} \rightarrow \varepsilon$
  - $ST \rightarrow \mathrm{bb}$

Contoh 1: Review

- Bahasa berupa himpunan string-string yang dibentuk dari alfabet $\Sigma = \{a, b\}$ di mana frekuensi kemunculan simbol 'a' sama dengan simbol 'b'
- $L = \{w \in \{a, b\}^*: \#_a w = \#_b w\}$
- Rule CFG untuk L adalah:

$$S \rightarrow aSb \qquad S \rightarrow SS$$

$$S \rightarrow bSa \qquad S \rightarrow \varepsilon$$

- String "ba", "aabb", dan "abbaba" dibentuk oleh CFG dengan proses derivasi sebagai berikut:

$S \Rightarrow bSa \Rightarrow ba$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

$S \Rightarrow SS \Rightarrow aSbbSa \Rightarrow abbaSba \Rightarrow abbaba$

Contoh 2: Review

L = $A^nB^n = \{a^n b^n : n \geq 0\}$.

- $S \rightarrow \varepsilon$
- $S \rightarrow aSb$

Language Balanced Parentheses

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

Rekursif

- *Rule* pada grammar G rekursif jika dan hanya jika berbentuk $X \rightarrow w_1 Y w_2$ di mana $Y =>_G^* w_3 X w_4$ dan $w_1, w_2, w_3, w_4 \in V^*$
- *Recursive rule* memungkinkan *finite* grammar membentuk *infinite* string
- Grammar G disebut rekursif jika dan hanya jika mengandung sekurang-kurangnya satu *recursive rule*.
- Setiap CFG memiliki sifat rekursif,
  - **Kecuali jika L(G) adalah bahasa berhingga!**
- Contoh: *rule* $S \rightarrow aSb$ dan $S \rightarrow SS$ rekursif
- Periksalah: *rule* $S \rightarrow (S) \mid (T)$ dan $T \rightarrow (S)$
- Q: Apakah grammar reguler bersifat rekursif?

Self-Embedding

- *Rule* pada grammar G *self-embedding* jika dan hanya jika berbentuk $X \rightarrow w_1Yw_2$ di mana $Y =>_G^* w_3Xw_4$ dan $w_1w_3, w_2w_4 \in \Sigma^+$
- Grammar G disebut *self-embedding* jika dan hanya jika mengandung paling tidak satu *self-embedding rule*.
  - Pada contoh 1, *rule* $S \rightarrow \mathrm{a}S\mathrm{b}$ *self-embedding*.
- **Jika G tidak *self-embedding*, maka $L(G)$ reguler.**
  - Q: Jika G *self-embedding, mungkinkah* $L(G)$ *reguler?* Contoh:

    $$G_1 = (\{S, \mathrm{a}\}, \{\mathrm{a}\}, \{S \rightarrow \varepsilon, \quad S \rightarrow \mathrm{a}, \quad S \rightarrow \mathrm{a}S\mathrm{a}\}, S)$$

Contoh-contoh Lain

$S \to \mathtt{a}S\mathtt{a}$	self-embedding
$S \to \mathtt{a}S$	recursive tapi tidak self-embedding
$S \to \mathtt{a}T$	
$T \to S\mathtt{a}$	self-embedding

Merancang CFG

- Jika string dalam *L* terdiri dari dua *region*, di mana terdapat relasi antar *region*, maka kedua *region* tersebut harus dibentuk secara bersamaan. A^nB^n
- Untuk membentuk string dengan multiple *region* yang urutannya tetap tetapi antar *region* tidak terdapat relasi, gunakan *rule* berbentuk $A \rightarrow BCD\ldots$

  Simbol non-terminal B, C, D merujuk pada *region*
- Untuk membentuk string dengan dua *region* yang urutannya tetap dan terdapat relasi antara keduanya, mulai dari sisi terluar (karakter pertama dan terakhir) string menuju ke tengah. $A \rightarrow \mathtt{a}A\mathtt{b}$

Konkatenasi Bahasa yang Independent

Let $L = \{a^n b^n c^m : n, m \geq 0\}$.

Keberadaan c^m *independent* terhadap $a^n b^n$, sehingga pembuatan bahasa L bisa dibagi menjadi 2 porsi yang terpisah dan akan dikokantenasi.

$G = (\{S, N, C, \text{a}, \text{b}, \text{c}\}, \{\text{a}, \text{b}, \text{c}\}, R, S\}$ where:

$$
\begin{aligned}
R = \{\ & S \rightarrow NC \\
& N \rightarrow \text{a}N\text{b} \\
& N \rightarrow \varepsilon \\
& C \rightarrow \text{c}C \\
& C \rightarrow \varepsilon\ \}.
\end{aligned}
$$

Unreachable/Nonproductive Rule

- Perhatikan grammar berikut

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

  $R = \{$

  $$S \to AB \mid AC \qquad C \to \mathrm{b}C\mathrm{a}$$
  $$A \to \mathrm{a}Ab \mid \varepsilon \qquad D \to AB$$
  $$B \to \mathrm{b}A$$

  $\}$
- 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

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

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 $X \to \alpha A \beta$

   (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*

Pembuktian Kebenaran CFG

- Diberikan CFG G dan bahasa L, bagaimana caranya membuktikan bahwa benar $L(G) = L$?
- Perlu dibuktikan:
  1. G menolak semua string pada L'

     Dengan kata lain, G hanya menerima string pada L
  2. G membentuk semua string pada L

- **Pembuktian 1** dengan menerapkan proses pembentukan string oleh G melalui serangkaian *loop*.
- **Pembuktian 2** dilakukan dengan cara induksi berdasarkan panjang string yang dibentuk.

Pembuktian Kebenaran CFG

Pembuktian 1 (detil)

- Definisikan w_s sebagai *working string*, beri harga awal S.
- Sampai tidak ada simbol non terminal tersisa pada w_s, terapkan sejumlah rule.
- Definisikan *loop invariant* I dan tunjukkan bahwa:
  - I bernilai *true* ketika *loop* dimulai
  - I dipertahankan setiap langkah berharga *true* melalui penerapan *rule* pada *loop*
  - $\mathrm{I} \wedge (w_s$ hanya berisi simbol terminal$) \rightarrow w_s \in L$

Contoh 1

- $L = \{w \in \{a, b\}^*: \#_a w = \#_b w\}$
- Rule CFG untuk L adalah:

  (1) $S \rightarrow aSb$ (3) $S \rightarrow SS$

  (2) $S \rightarrow bSa$ (4) $S \rightarrow \varepsilon$
- Untuk membantu pembuktian, didefinisikan notasi $\Delta w = \#_a w - \#_b w$
- Loop invariant I = $(\Delta w_s == 0)$
- Pembuktian 1

  Buktikan bahwa Δw selalu bernilai 0

Contoh 1

Pembuktian 1

- Melalui derivasi, diperoleh $st \in \{a, b, S\}^* \wedge \Delta w_s = 0$.
- I true ketika $w_s = S$ karena $\Delta w_s = 0$
- I true setelah penerapan rule. Penerapan rule (1) dan (2) berarti menambahkan masing-masing 1 simbol a dan b, sehingga Δw_s tetap bernilai 0. Penerapan rule (3) tidak menambahkan simbol a maupun b.
- Jika I true dan w_s mengandung hanya simbol terminal, maka $w_s \in L$. Dalam hal ini, ditunjukkan bahwa w_s hanya mengandung simbol a dan b, serta $\Delta w_s = 0$

Contoh 1

Pembuktian 2

- Karena setiap string w pada L harus mengandung simbol a dan b dengan jumlah kemunculan yang sama, maka $|w|$ genap.
- Base case: $|w| = 0$ dan $w = \varepsilon$

  Terbukti benar dengan menerapkan rule (4) pada S
- Hipotesis induksi:

  Jika setiap string pada L dengan panjang $\leq k$, (k genap) dapat dibentuk dengan G, maka string pada L dengan panjang $k + 2$ juga dapat dibentuk dengan G

Contoh 1

Pembuktian 2 (lanjutan)

- Jika $w \in L$, $|w| = k + 2$, $x \in \{a, b\}^*$, dan $|x| = k$, maka $w = axb$ atau $w = bxa$ atau $w = axa$ atau $w = bxb$
- Kasus 1 ($w = axb$ atau $w = bxa$)

  Penerapan rule (1) dan (2)
- Kasus 2 ($w = axa$ atau $w = bxb$)

  Semua string w dengan bentuk pada kasus 2 sekurang-kurangnya memiliki panjang 4. Dapat dibentuk dengan menerapkan rule (3)

Parse Tree

- Ingat bahwa:
  - $x \Rightarrow_G y$ adalah relasi *derive in-one-step* string x menjadi y (dimana $x, y \in V^*$) melalui aplikasi suatu rule dalam R_G.
  - Deretan relasi $S \Rightarrow_G x_1 \Rightarrow_G x_2 \ldots \Rightarrow w$ (atau disingkat $S \Rightarrow_G^* w$) 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 $X \rightarrow x_1, x_2, \ldots, x_n$.

$S \rightarrow NP\ VP$

$NP \rightarrow \texttt{the}\ Nominal \mid \texttt{a}\ Nominal \mid Nominal \mid ProperNoun \mid NP\ PP$

$Nominal \rightarrow N \mid Adjs\ N$

$N \rightarrow \texttt{cat} \mid \texttt{dogs} \mid \texttt{bear} \mid \texttt{girl} \mid \texttt{chocolate} \mid \texttt{rifle}$

$ProperNoun \rightarrow \texttt{Chris} \mid \texttt{Fluffy}$

$Adjs \rightarrow Adj\ Adjs \mid Adj$

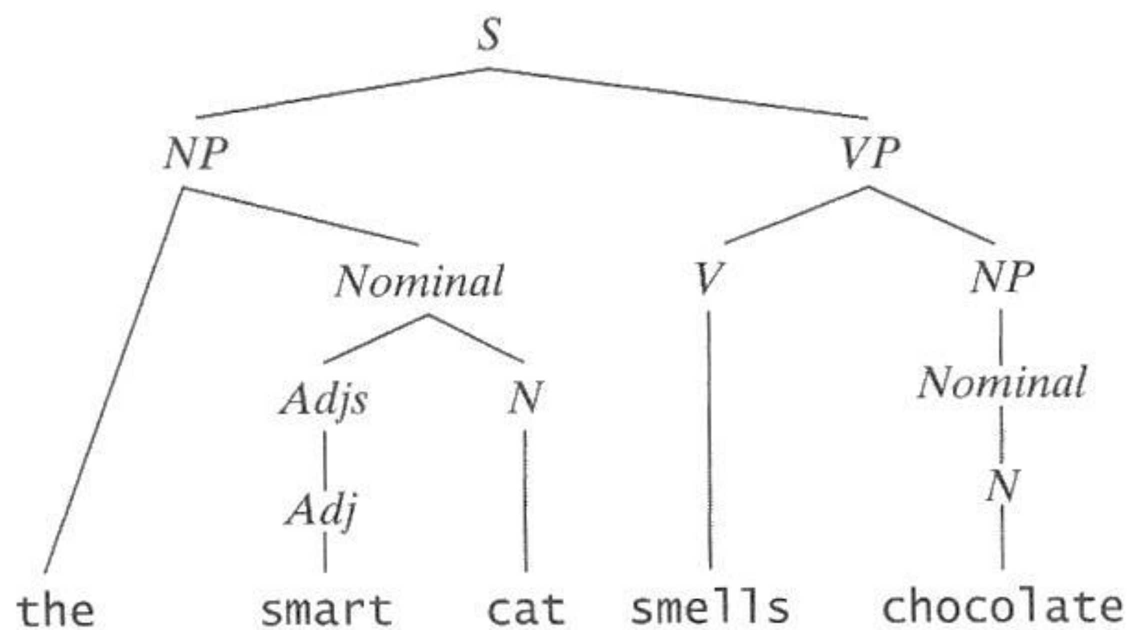
$Adj \rightarrow \texttt{young} \mid \texttt{older} \mid \texttt{smart}$

$VP \rightarrow V \mid V\ NP \mid VP\ PP$

$V \rightarrow \texttt{like} \mid \texttt{likes} \mid \texttt{thinks} \mid \texttt{shot} \mid \texttt{smells}$

$PP \rightarrow Prep\ NP$

$Prep \rightarrow \texttt{with}$



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