BAB V
CONTEXT FREE GRAMMAR DAN PUSH DOWN AUTOMATA

TUJUAN PRAKTIKUM

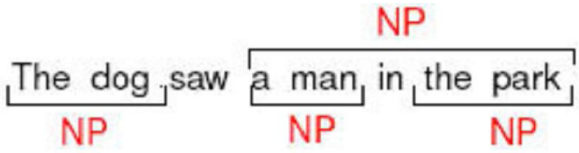
1. Memahami CFG dan PDA
2. Memahami Context Free Grammar
3. Memahami Push Down Automata
4. Mengerti Tentang Operasi - Operasi yang dilakukan

TEORI PENUNJANG

5.1 Context Free Grammar (CFG)

Terinspirasi dari bahasa natural manusia, ilmuwan-ilmuwan ilmu komputer yang mengembangkan bahasa pemrograman turut serta memberikan tata bahasa (pemrograman) secara formal. Tata bahasa ini diciptakan secara bebas-konteks dan disebut CFG (*Context Free Grammar).* Hasilnya, dengan pendekatan formal ini, kompiler suatu bahasa pemrograman dapat dibuat lebih mudah dan menghindari ambiguitas ketika parsing bahasa tersebut. Contoh desain CFG untuk parser, misal : B -> BB | (B) | e untuk mengenali bahasa dengan hanya tanda kurung {'(',')'} sebagai terminal-nya. Proses parsing adalah proses pembacaan untai dalam bahasa sesuai CFG tertentu, proses ini harus mematuhi aturan produksi dalam CFG tersebut.

Secara formal, CFG didefinisikan[2] : **CFG G = (V,T,P,S)**, dimana **V** adalah daftar variabel produksi **T**, adalah daftar simbol atau terminal yang dipakai dalam CFG **P**, adalah aturan produksi CFG **S**, adalah variabel start aturan produksi CFG dapat 'dinormalkan' dengan pola tersendiri supaya tidak ambigu dan lebih sederhana, meskipun normalisasi CFG kadang membuat aturan produksi menjadi lebih banyak dari sebelumnya. Teknik normalisasi yang digunakan dalam makalah ini adalah CNF (*Chomsky Normal Form).*



Gambar 5.1. Gambaran parsing bahasa natural (Inggris)

Contoh desain CNF dari bahasa CFG, semisal CFGberikut:

S -> aA | bB (1)

A-> Baalba

B -> bAAlab

CFG (1) tersebut ekivalen dengan CFG dibawah ini,dimana symbol terminal memiliki variabel produksi tersendiri:

S -> DA | EB (2)

A -> BDD | ED

B -> EAA | DE

D -> a

E -> b

CNF yang dihasilkan dari CFG (2) diatas ialah:

S -> DA | EB (3)

A -> BF | ED

B -> EH | DE

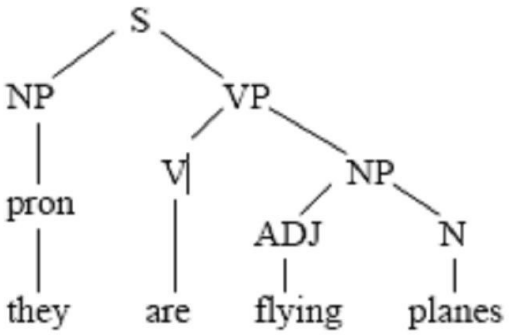
F -> DD

H -> AA

D -> a

E -> b

Setelah terbentuk CFG yang telah dinormalkan secara CNF, dalam implementasi parsing, terdapat algoritma yang berguna untuk menentukan apakah suatu untai 'valid', atau dapat diciptakan dari aturan-aturan CFG yang ada. Salah satu algoritma yang dapat dipakai adalah Algoritma Cocke-Younger-Kasami (CYK). Algoritma ini menyelesaikan masalah analisa kembali sebuah sub-untai yang sama karena seharusnya analisa sub-untai independen terhadap parsing sub-untai yang diparsing setelahnya. Dengan Program Dinamis, independensi yang diinginkan dapat dicapai ketika parsing. Algoritma CYK termasuk dalam bidang Program Dinamis karena algoritma ini membangun tabel status dua dimensi ketika parsing dimana penentuan parsing selanjutnya diturunkan atau dihasilkan dari parsing sebelumnya, hingga akhir untai. Selain untuk mengetahui validitas untai dalam suatu CFG, algoritma CYK yang dimodifikasi dapat dipergunakan pula untuk membangun pohon parsing.



Gambar 5.2 Pohon parsing yang terbentuk dari sebuah bahasa natural

Contoh 1 :

Diketahui grammar $G_1 = \{I \rightarrow H \mid I\,H \mid IA,\ H \rightarrow a \mid b \mid c \mid ... \mid z,\ A \rightarrow 0 \mid 1 \mid 2 \mid ... \mid 9\}$

dengan I adalah simbol awal. Berikut ini kedua cara analisa sintaks untuk kalimat x23b.

cara 1 (derivasi)

$I \Rightarrow IH$

$\Rightarrow IAH$

$\Rightarrow IAAH$

$\Rightarrow HAAH$

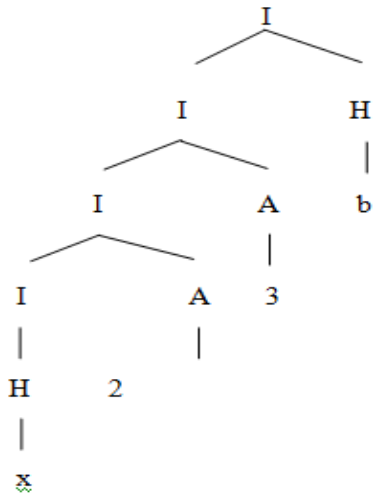
$\Rightarrow xAAH$

$\Rightarrow x2AH$

$\Rightarrow x23H$

$\Rightarrow x23b$

cara 2 (parsing)

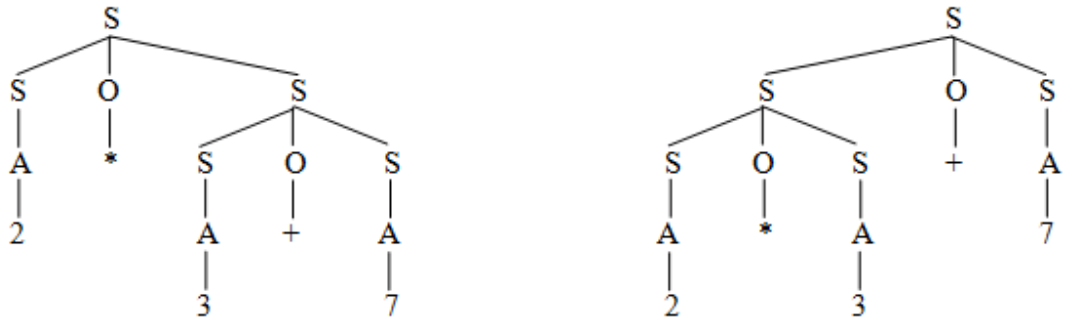


Sebuah kalimat dapat saja mempunyai lebih dari satu pohon.

Contoh 2 :

Diketahui grammar $G_2 = \{S \rightarrow SOS \mid A,\ O \rightarrow * \mid +,\ A \rightarrow 0 \mid 1 \mid 2 \mid ... \mid 9\}$

Kalimat : 2*3+7 mempunyai dua pohon sintaks berikut :



Sebuah kalimat yang mempunyai lebih dari satu pohon sintaks disebut *kalimat ambigu* (*ambiguous*). Grammar yang menghasilkan paling sedikit sebuah kalimat ambigu disebut *grammar ambigu*.

5.2 Push Down Automata (PDA)

PDA adalah mesin otomata dari TBBK yang diimplementasikan dengan stack sehingga hanya terdapat operasi "push" dan "pop" Stack (tumpukan) adalah suatu struktur data yang menggunakan prinsip LIFO (Last In First Out). Sebuah stack selalu memiliki top of stack dan elemen-elemen stack itu yang akan masuk ke dalam stack dengan method "push" dan akan keluar dari stack dengan method "pop".

- Definisi : PDA adalah pasangan 7 tuple $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, A)$, dimana :

  Q : himpunan hingga stata, Σ : alfabet input, Γ : alfabet *stack*, $q_0 \in Q$: stata awal,

  $Z_0 \in \Gamma$: simbol awal *stack*, $A \subseteq Q$: himpunan stata penerima,

  fungsi transisi $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$ (himpunan bagian dari $Q \times \Gamma^*$)
- Untuk stata $q \in Q$, simbol input $a \in \Sigma$, dan simbol *stack* $X \in \Gamma$, $\delta(q, a, X) = (p, \alpha)$ berarti : PDA bertransisi ke stata p dan mengganti X pada stack dengan string α.
- Konfigurasi PDA pada suatu saat dinyatakan sebagai triple (q, x, α), dimana :

  $q \in Q$: stata pada saat tersebut, $x \in \Sigma^*$: bagian string input yang belum dibaca, dan $\alpha \in \Gamma^*$: string yang menyatakan isi *stack* dengan karakter terkiri menyatakan *top of stack*.
- Misalkan $(p, ay, X\beta)$ adalah sebuah konfigurasi, dimana : $a \in \Sigma$, $y \in \Sigma^*$, $X \in \Gamma$, dan $\beta \in \Gamma^*$. Misalkan pula $\delta(p, a, X) = (q, \gamma)$ untuk $q \in Q$ dan $\gamma \in \Gamma^*$. Dapat kita tuliskan bahwa : $(p, ay, X\beta) \Rightarrow (q, y, \gamma\beta)$.

Sebuah PDA dinyatakan dengan :

Q = himpunan state

Σ = himpunan simbol input

T = simbol stack

Δ = fungsi transisi

S = state awal

F = state akhir

Z = top of stack

PDA memiliki 2 jenis transisi, yaitu Δ yang merima simbol input, simbol top of stack, dan state. Setiap pilihan terdiri dari state berikutnya dan simbol- simbol. Penggantian isi stack dilakukan dengan opersi push dan pop. Jenis transisi yang kedua adalah transisi ε. Transisi ε tidak melakukan pembacaan input namun hanya menerima simbol top of stack dan state. Transisi ini memungkinkan PDA untuk memanipulasi isi stack dan berpindah antar state tanpa membaca input.

Contoh 14 (PDA Deterministik):

PDA M = (Q, Σ, Γ, q_0, Z_0, δ, A) pengenal palindrome L = $\{xcx^T \mid x \in (a|b)^*\}$, dimana x^T adalah cermin(x), mempunyai tuple : Q = $\{q_0, q_1, q_2\}$, A = $\{q_2\}$, Σ = {a, b, c}, Γ = {a, b, Z_0}, dan fungsi transisi δ terdefinisi melalui tabel berikut :

No.	Stata	Input	*TopStack*	Hasil
1	q_0	a	Z_0	(q_0, aZ_0)
2	q_0	b	Z_0	(q_0, bZ_0)
3	q_0	a	a	(q_0, aa)
4	q_0	b	a	(q_0, ba)
5	q_0	a	b	(q_0, ab)
6	q_0	b	b	(q_0, bb)

No.	Stata	Input	*TopStack*	Hasil
7	q_0	c	Z_0	(q_1, Z_0)
8	q_0	c	a	(q_1, a)
9	q_0	c	b	(q_1, b)
10	q_1	a	a	(q_1, ε)
11	q_1	b	b	(q_1, ε)
12	q_1	ε	Z_0	(q_2, ε)

Sebagai contoh, perhatikan bahwa fungsi transisi No. 1 dapat dinyatakan sebagai : $\delta(q_0, a, Z_0) = (q_0, aZ_0)$. Pada tabel transisi tersebut terlihat bahwa pada stata q_0 PDA akan melakukan PUSH jika mendapat input a atau b dan melakukan transisi stata ke stata q_1 jika mendapat input c. Pada stata q_1 PDA akan melakukan POP.

Berikut ini pengenalan dua string oleh PDA di atas :

1. abcba : $(q_0, abcba, Z_0) \Rightarrow (q_0, bcba, aZ_0)$ (1)
   $\Rightarrow (q_0, cba, baZ_0)$ (4)
   $\Rightarrow (q_1, ba, baZ_0)$ (9)
   $\Rightarrow (q_1, a, aZ_0)$ (11)
   $\Rightarrow (q_1, \varepsilon, Z_0)$ (10)
   $\Rightarrow (q_2, \varepsilon, Z_0)$ (12) (*diterima*)

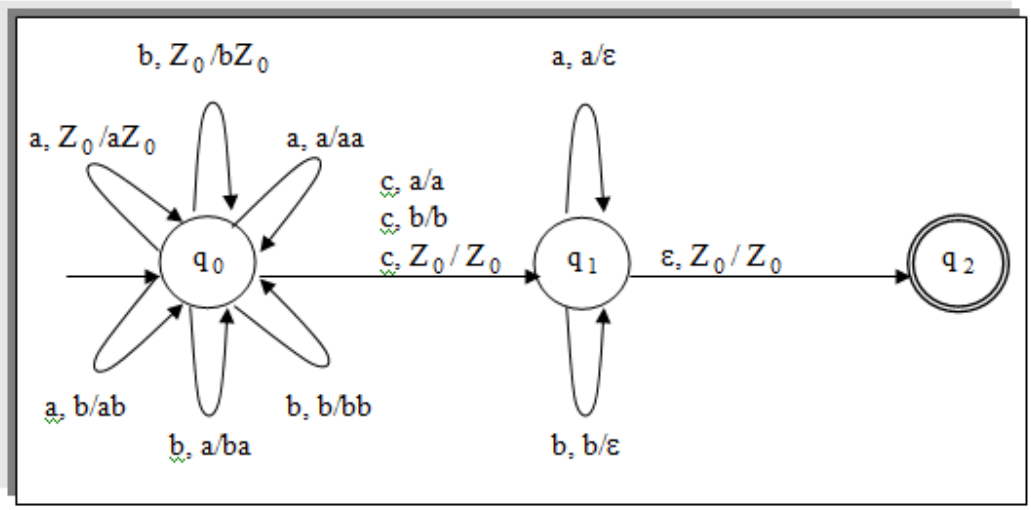
2. acb : $(q_0, acb, Z_0) \Rightarrow (q_0, cb, aZ_0)$ (1)
   $\Rightarrow (q_1, b, aZ_0)$ (8), (*crash → ditolak*)

3. ab : $(q_0, ab, Z_0) \Rightarrow (q_0, b, aZ_0)$ (1)
   $\Rightarrow (q_0, \varepsilon, baZ_0)$ (4) (*crash → ditolak*)

Penerimaan dan penolakan tiga string di atas dapat dijelaskan sebagai berikut :

1. string abcba diterima karena *tracing* sampai di stata penerima (q_2) dan string "abcba" selesai dibaca (string yang belum dibaca = ε)
2. string acb ditolak karena konfigurasi akhir (q_1, b, a Z_0) sedangkan fungsi transisi $\delta(q_1, b, a)$ tidak terdefinsi
3. string ab ditolak karena konfigurasi akhir $(q_0, \varepsilon, baZ_0)$ sedangkan fungsi transisi $\delta(q_0, \varepsilon, b)$ tidak terdefinsi

Ilustrasi graf fungsi transisi PDA di atas ditunjukkan melalui gambar berikut :



- Notasi $(p, ay, X\beta) \Rightarrow (q, y, \gamma\beta)$ dapat diperluas menjadi : $(p, x, \alpha) \Rightarrow^* (q, y, \beta)$, yang berarti konfigurasi (q, y, β) dicapai melalui sejumlah (0 atau lebih) transisi.
- Ada dua cara penerimaan sebuah kalimat oleh PDA, yang masing-masing terlihat dari konfigurasi akhir, sebagaimana penjelasan berikut :

Jika M = (Q, Σ, Γ, q_0, Z_0, δ, A) adalah PDA dan x ∈Σ*, maka x *diterima dengan stata akhir* (*accepted by final state*) oleh PDA M jika : (q_0, x, Z_0) ⇒* (q, ε, α) untuk α ∈ Γ * dan q ∈ A. x *diterima dengan stack hampa* (*accepted by empty stack*) oleh PDA M jika : (q_0, x, Z_0) ⇒* (q, ε, ε) untuk q ∈ Q.

Contoh 15 (PDA Non-Deterministik):

PDA M = (Q, Σ, Γ, q_0, Z_0, δ, A) pengenal palindrome L = {xx^T | x ∈ (a|b)*} mempunyai komponen tuple berikut : Q = {q_0, q_1, q_2}, A = {q_2}, Σ = {a, b}, Γ = {a, b, Z_0}, dan fungsi transisi δ terdefinisi melalui tabel berikut :

No.	St.	In.	*TS*	Hasil
1	q_0	a	Z_0	(q_0, aZ_0),(q_1, Z_0)
2	q_0	b	Z_0	(q_0, bZ_0),(q_1, Z_0)
3	q_0	a	a	(q_0, aa),(q_1, a)
4	q_0	b	a	(q_0, ba),(q_1, a)
5	q_0	a	b	(q_0, ab),(q_1, b)
6	q_0	b	b	(q_0, bb),(q_1, b)

No.	St.	In.	*TS*	Hasil
7	q_0	ε	Z_0	(q_1, Z_0)
8	q_0	ε	a	(q_1, a)
9	q_0	ε	b	(q_1, b)
10	q_1	a	a	(q_1, ε)
11	q_1	b	b	(q_1, ε)
12	q_1	ε	Z_0	(q_2, ε)

Pada tabel transisi tersebut terlihat bahwa pada stata q_0 PDA akan melakukan PUSH jika mendapat input a atau b dan melakukan transisi stata ke stata q_1 jika mendapat input ε. Pada stata q_1 PDA akan melakukan POP. Contoh 14 dan 15 menunjukkan bahwa PDA dapat dinyatakan sebagai mesin PUSH-POP.

Berikut ini pengenalan string “baab” oleh PDA di atas :

1. (q_0, baab, Z_0) ⇒ (q_0, aab, bZ_0) (2 kiri)
   ⇒ (q_0, ab, abZ_0) (5 kiri)
   ⇒ (q_1, ab, abZ_0) (3 kanan)
   ⇒ (q_1, b, bZ_0) (11)
   ⇒ (q_1, ε, Z_0) (10)
   ⇒ (q_2, ε, Z_0) (12) (*diterima*)

2. $(q_0, baab, Z_0)$ $\Rightarrow (q_1, baab, Z_0)$ (2 kanan) (*crash* $\rightarrow$ *ditolak*)

3. $(q_0, baab, Z_0)$ $\Rightarrow (q_0, aab, bZ_0)$ (2 kiri)
   $\Rightarrow (q_0, ab, abZ_0)$ (5 kiri)
   $\Rightarrow (q_0, b, aabZ_0)$ (3 kiri)
   $\Rightarrow (q_1, b, aabZ_0)$ (4 kanan) (*crash* $\rightarrow$ *ditolak*)

4. $(q_0, baab, Z_0)$ $\Rightarrow (q_0, aab, bZ_0)$ (2 kiri)
   $\Rightarrow (q_0, ab, abZ_0)$ (5 kiri)
   $\Rightarrow (q_0, b, aabZ_0)$ (3 kiri)
   $\Rightarrow (q_0, \varepsilon, baabZ_0)$ (4 kiri)
   $\Rightarrow (q_1, \varepsilon, baabZ_0)$ (9) (*crash* $\rightarrow$ *ditolak*)