BAB IV
ANALISA SINTAKS

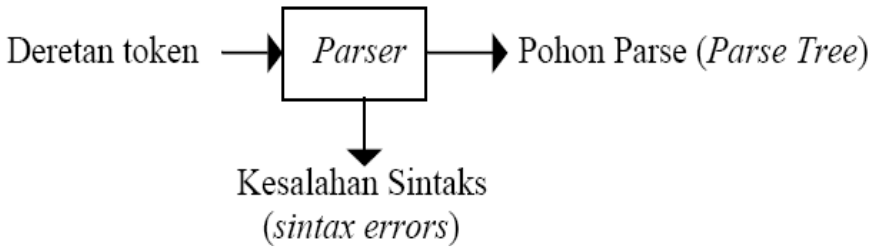
TUJUAN PRAKTIKUM

1) Memahami dan mengerti tugas analisa sintaks.
2) Memahami dan mengerti predictive parsing.
3) Memahami dan mengerti parsing Table M.

TEORI PENUNJANG

4.1. Posisi Parser dalam Kompilator

Posisi Penganalisa Sintaks (*Parser*) dalam proses kompilasi adalah sebagai berikut :



Gambar 4.1 : Skema Parser

- Deretan token : dihasilkan oleh Penganalisa Leksikal (*Scanner*)
- Pohon parse : suatu pohon dimana akarnya (*root*) adalah *simbol awal grammar* (*starting symbol*), setiap node dalam (*inner node*) adalah simbol nonterminal, dan daunnya (*leaf*) dibaca dari kiri ke kanan adalah *deretan token masukan*. Pohon parse ini dibentuk berdasarkan aturan grammar yang ditetapkan untuk *parser*.
- Kesalahan sintaks : terjadi jika pola deretan token tidak memenuhi ketentuan pola yang telah ditentukan grammar untuk *parser*.

Grammar yang dipilih untuk *scanner* adalah *Regular Grammar* (*RG*) sedangkan untuk *parser* adalah *Grammar Context Free* (*CFG*). Penting diketahui perbedaan *cara pandang RG* dengan *CFG* terhadap sebuah token yang mengalir antara *scanner* dan *parser*. Bagi RG (*scanner*) sebuah token (kecuali *reserve word*) adalah *sebuah kalimat* dimana setiap karakter pembentuk token tersebut adalah *simbol terminal*. Sebaliknya bagi CFG (*parser*) sebuah token adalah *sebuah simbol terminal* dimana sederetan tertentu token akan membentuk *sebuah kalimat*.

4.2. Review Hal-hal Penting dari CFG

a. Pola umum CFG : $A \rightarrow \alpha$, $A \in V_N$, $\alpha \in (V_N \mid V_T)^*$

b. Sifat *ambigu* (*ambiguity*) :

Sebuah kalimat adalah *ambigu* jika terdapat lebih dari satu pohon sintaks yang dapat dibentuk oleh kalimat tersebut. Secara gramatikal kalimat ambigu dihasilkan oleh *grammar ambigu* yaitu grammar yang mengandung *beberapa* produksi dengan *ruas kiri yang* sama sedangkan dua atau lebih ruas kanan-nya mempunyai *string terkiri* (*prefix*) yang sama. Contoh :

$S \rightarrow$ if E then S | if E then S else S,

dengan S : *statement* dan E : *expression*, mempunya dua produksi dengan ruas kiri sama (S) sedangkan kedua ruas kanannya dimulai dengan string sama (if E then S).

Grammar ambigu dapat diperbaiki dengan metoda *faktorisasi kiri* (*left factorization*). *Prefix* dari produksi di atas adalah *sentensial* if E then S sehingga faktorisasi akan menghasilkan :

$S \rightarrow$ if E then S T, $T \rightarrow \varepsilon$ | else S {ε : simbol hampa}

c. Sifat rekursi kiri (*left recursion*) :

Sebuah grammar dikatakan bersifat rekursi kiri jika untuk sebuah simbol nonterminal A terdapat *derivasi non hampa* $A \Rightarrow \ldots \Rightarrow A\alpha$. Produksi berbentuk $A \rightarrow A\alpha$ disebut produksi yang bersifat *immediate left recursion*.

Rekursi kiri dapat dieliminir dengan transformasi berikut :

$A \rightarrow A\alpha \mid \beta$ transformasi menjadi : $A \rightarrow \beta R$, $R \rightarrow \alpha R \mid \varepsilon$

Transformasi ini dapat diperluas sehingga :

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid ... \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid ... \mid \beta_n$$

bertransformasi menjadi :

$$A \rightarrow \beta_1 R \mid \beta_2 R \mid ... \mid \beta_n R, \quad R \rightarrow \alpha_1 R \mid \alpha_2 R \mid .. \mid \alpha_n R \mid \varepsilon$$

Contoh 1 : Diketahui : $E \rightarrow E + T \mid T,\ T \rightarrow T * F \mid F,\ F \rightarrow (E) \mid I$

yang jelas mengandung *immediate left recursion* untuk simbol E dan T.

Transformasi menghasilkan :

$$E \rightarrow TR_E, R_E \rightarrow +TR_E \mid \varepsilon,\ \ T \rightarrow FR_T, R_T \rightarrow *FR_T \mid \varepsilon,\ \ F \rightarrow (E) \mid I$$

Prosedur transformasi di atas dituangkan dalam algoritma berikut :

Algoritma Rekursi_Kiri

1. *Rename* semua nonterminal menjadi $A_1, A_2, ..., A_n$
2. for i = 1 to n do begin

2.a. for j = 1 to i-1 do begin

ganti setiap produksi berbentuk $A_i \rightarrow A_j \gamma$

dengan produksi-produksi : $A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid ... \mid \delta_k \gamma$

dimana : $A_j \rightarrow \delta_1 \mid \delta_2 \mid ... \mid \delta_k$ produksi-A_j saat iterasi ini

end

2.b. eliminasi semua *immediate left recursion* produksi-A_i

end

Contoh 2 : Diketahui : $S \rightarrow Aa \mid b,\ A \rightarrow Ac \mid S\ d \mid \varepsilon$

Algoritma Rekursi_Kiri akan digunakan terhadap himpunan produksi ini.

Langkah 1 : $A_1 := S,\ A_2 := A$ sehingga produksi menjadi

$$A_1 \rightarrow A_2 a \mid b,\ A_2 \rightarrow A_2 c \mid A_1 d \mid \varepsilon$$

Saat i = 1 *inner loop* tidak dipenuhi karena (j =1) > (i-1= 0), maka program masuk ke (2.b) untuk A_1. Tetapi A_1 tidak bersifat *immediate left recursion*. Jadi saat i = 1 program tidak melakukan apapun.

Saat i = 2,

(2.a) j = 1 : ganti $A_2 \rightarrow A_1 d$ dengan $A_2 \rightarrow A_2 ad \mid bd$

(2.b) $A_2 \rightarrow A_2 c \mid A_2 ad \mid bd \mid \varepsilon$ adalah *immediate left recursion*, sehingga diperoleh transformasinya :

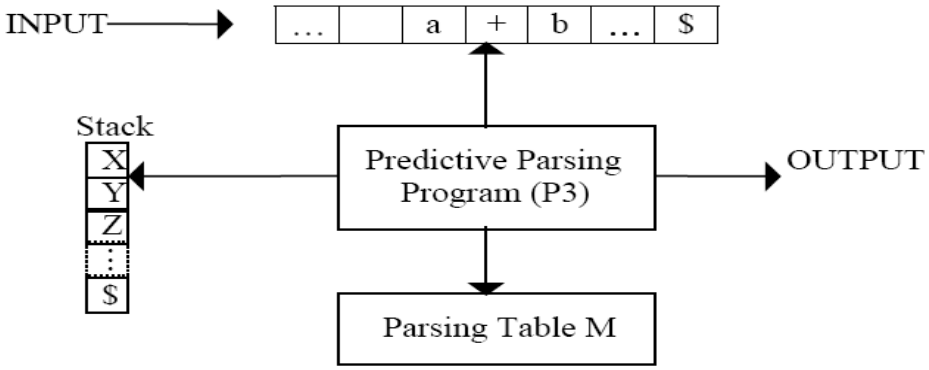
$$A_2 \rightarrow bdR_A \mid R_A,\ R_A \rightarrow c\,R_A \mid adR_A \mid \varepsilon$$

Hasilnya : $A_1 \rightarrow A_2 a \mid b,\ A_2 \rightarrow bdR_A \mid R_A,\ R_A \rightarrow c\,R_A \mid adR_A \mid \varepsilon$, atau :

$$S \rightarrow Aa \mid b,\ A \rightarrow bdR_A \mid R_A,\ R_A \rightarrow cR_A \mid adR_A \mid \varepsilon$$

4.3. Predictive Parser

Predictive Parser akan digunakan untuk mengimplementasikan Penganalisa Sintaks (*Parser*). Berikut ini adalah model dari *Predictive Parser*.



Gambar 4.2 : Skema Predictive Parser

Input : rangkaian token dan diakhiri dengan tanda $.

Stack : berisi simbol grammar (V_N atau V_T). Pada keadaan awal stack hanya berisi $ dan S (simbol awal).

Parsing Table M : array 2 dimensi M(A,a), dimana A adalah simbol nonterminal dan a adalah simbol terminal (token)

atau sibol $. Nilai M(A,a) adalah : sebuah produksi $A \rightarrow \alpha$ atau tanda-tanda kesalahan (keduanya akan dibahas kemudian)

Predictive Parsing Program (P3) : sebuah program yang mengendalikan *parser* berdasar-kan nilai A dan a.

Sifat dan tanggapan P3 terhadap simbol A (pada *stack*) dan a (pada *input*) :

1. Jika A = a = $: *parser* berhenti dan memberitahukan bahwa kerja *parser* telah selesai tanpa ditemukan kesalahan sintaks.
2. Jika A = a ≠ $: *parser* akan mengeluarkan A dari *stack*, dan selanjutnya membaca token berikutnya.
3. Jika $A \in V_T$, A ≠ a : terjadi kesalahan sintaks, dan selanjutnya akan dipanggil *routine penanganan kesalahan* (*error handler*).
4. Jika $A \in V_N$: program akan membaca tabel M(A,a). Jika M(A,a) berisi produksi $A \rightarrow$ UVW maka *parser* akan mengganti A dengan WVU (yaitu dengan U berada di puncak *stack*). Jika M(A,a) berisi tanda-tanda kesalahan maka *parser* akan memang- gil *Error_Handler routine*.

4.4. Parsing Table M

Parsing Table M dibentuk berdasarkan dua fungsi yang berhubungan dengan suatu tata bahasa. Kedua fungsi tersebut adalah *First*(X), $X \in (V_N \mid V_T)$ dan *Follow*(Y), $Y \in V_N$.

First(X) adalah himpunan *simbol terminal* yang merupakan simbol pertama dari X atau merupakan simbol pertama dari simbol-simbol yang dapat diturunkan dari X.

Follow(Y) adalah himpunan *simbol terminal* yang dapat muncul tepat di sebelah kanan Y melalui nol atau lebih derivasi.

Ketentuan selengkapnya tentang First(X) dan Follow(Y) adalah sebagai berikut:

a. First(X)

a1. Jika $X \in V_T$ maka First(X) = {X}

a2. Jika terdapat $X \rightarrow a\alpha$ maka $a \in$ First(X). Jika $X \rightarrow \varepsilon$ maka $\varepsilon \in$ First(X)

a3. Jika $X \rightarrow Y_1 Y_2 ... Y_k$ maka $First(Y_1) \subset First(X)$. Jika ternyata Y_1 dapat men-derivasi ε (sehingga $\varepsilon \in First(Y_1)$) maka $First(Y_2)$ juga *subset* dari First(X). Jelaslah jika semua $First(Y_i)$ mengandung ε , i = 1, 2, ..., n, maka semua elemen $First(Y_i)$ adalah juga elemen First(X).

b. Follow(X)

b1. Jika X = S = simbol awal maka $\$ \in Follow(S)$

b2. Jika $X \rightarrow \alpha Y \beta$, $\beta \neq \varepsilon$, maka $\{First(\beta) - \{\varepsilon\}\} \subset Follow(Y)$

b3. Jika 1. $X \rightarrow \alpha Y$ atau 2. $X \rightarrow \alpha Y \beta$ dimana $\varepsilon \in First(\beta)$ maka $Follow(X) \subset Follow(Y)$

Contoh :

Diketahui himpunan produksi :

1. $E \rightarrow TE'$, 2. $E' \rightarrow +TE' \mid \varepsilon$, 3. $T \rightarrow FT'$, 4. $T' \rightarrow *FT' \mid \varepsilon$, 5. $F \rightarrow (E) \mid id$

Fisrt : ◊ dari (5) dengan aturan (a2) : First(F) = {(, id}

◊ dari (3) dengan aturan (a3) : First(T) = First(F)

dari (1) dengan aturan (a3) : First(E) = Fisrt(T)

sehingga : First(E) = Fisrt(T) = First(F) = {(, id}

◊ dari(2) dengan aturan (a2) : First(E') = {+, ε}

◊ dari (4) dengan aturan (a2) : First(T') = {*, ε}

Follow : ◊ dari(1) dengan aturan (b1) : $\$ \in Follow(E)$ karena E = simbol awal,

dari (5) dengan aturan (b2) dan (a1) :) ∈ Follow(E)

sehingga : Follow(E) = {$,)}

◊ dari(1) dengan aturan (b3.1) $X \rightarrow \alpha Y$: $Follow(E) \subset Follow(E')$

sehingga Follow(E') = {$,)}

◊ dari(1) (dan (2)) dengan aturan (b2) : $\{First(E') - \{\varepsilon\}\} = \{+\} \subset Follow(T)$

dari(1) dan aturan (b3.2) $X \rightarrow \alpha Y \beta$, $\alpha = \varepsilon$: $Follow(E) = \{\$,)\} \subset Follow(T)$

sehingga : Follow(T) = {$,), +}

◊ dari(3) dengan aturan(b3.1) : $X \rightarrow \alpha Y$: $Follow(T) \subset Follow(T')$

sehingga : Follow(T') = {$,), +}

◊ dari(4) dengan aturan (b2) : {First(T') - {ε}} = {*} ⊂ Follow(F)

dari(3) dengan aturan(b3.2) $X \rightarrow \alpha Y \beta$, $\alpha = \varepsilon$: Follow(T) ⊂ Follow(F)

sehingga : Follow(F) = {$,), +,*}

Singkatnya : First(E) = Fisrt(T) = First(F) = {(, id}

First(E') = {+, ε}

First(T') = {*, ε}

Follow(E) = Follow(E') ={$,)}

Follow(T) = Follow(T') = {$,), +,*}

Follow(F) = {$,), +,*}

Dengan kedua fungsi First(X) dan Follow(X) di atas maka Parsing Table M disusun melalui algoritma berikut :

Algoritma Parsing_Table

1. for setiap produksi $A \rightarrow \alpha$ do begin

1a. for setiap $a \in$ First(α) do M(A, a) = $A \rightarrow \alpha$

1b. if $\varepsilon \in$ First(α) then

for setiap $b \in$ Follow(A) do M(A, b) = $A \rightarrow \alpha$

1c. if ($\varepsilon \in$ First(α)) and ($\$ \in$ Follow(A) then M(A, $) = $A \rightarrow \alpha$

end

2. if M(A, a) = blank then M(A, a) = error

Jika algoritma di atas diterapkan kepada grammar :

1. $E \rightarrow TE'$, 2. $E' \rightarrow +TE' \mid \varepsilon$, 3. $T \rightarrow FT'$, 4. $T' \rightarrow *FT' \mid \varepsilon$, 5. $F \rightarrow (E) \mid id$

Maka :

◊ $E \rightarrow TE'$

Karena : First(TE') = Fisrt(T) = {(, id} maka menurut (1a) :

M(E, () = M(E, id) = E → TE’

◊ E’→ +TE’

Karena : + ∈ FIRST(+TE’) maka menurut aturan (1a) : M(E’, +) = E’→ +TE’

◊ E’→ ε

Karena : ε ∈ FIRST(E’) dan {), $) ⊂ Follow(E’) maka menurut aturan (1b) :

M(E’,)) = M(E’, $) = E’→ ε

◊ T→ FT’

Karena : First(FT’) = First(F) = {(, id) maka menurut aturan (1a) :

M(T, () = M(T, id) = T→ FT’

◊ T’→ *FT’

Karena : * ∈ First(*FT’) maka menurut aturan (1a) : M(T’, *) = T’→ *FT’

◊ T’ → ε

Karena : ε ∈ First(T’) dan {+,), $) ⊂ Follow(T’) maka menurut aturan (1b) :

M(T’, +) = M(T’,)) = M(T’, $) = T’ → ε

◊ F → id

Karena : id ∈ First(F) maka menurut aturan (1a) : M(F, id) = F → id

◊ F → (E)

Karena : (∈ First(F) maka menurut aturan (1a) : M(F, () = F → (E)

Akhirnya diperoleh tabel berikut :

Non Terminal	Simbol Input					
	id	+	*	(	)	$
E	E → TE’	error	error	E → TE’	error	error
E’	error	E’→ +TE’	error	error	E’→ ε	E’→ ε
T	T→ FT’	error	error	T→ FT’	error	error
T’	error	T’ → ε	T’→ *FT’	error	T’→ ε	T’→ ε
F	F → id	error	error	F→ (E)	error	error

Berikut ini ditunjukkan tabulasi aksi yang dilakukan *predictive parser* berdasarkan *parser table M* di atas terhadap rangkaian token : id + id * id :

Stack	Input	Output	Keterangan
$E	id + id * id $		E = simbol awal, M(E, id) = E → TE' ganti E dengan TE', T di *top of stack*
$E'T	id + id * id $	E → TE'	M(T, id) = T→ FT' (ganti T dengan FT')
$E'T'F	id + id * id $	T→ FT'	M(F, id) = F → id (ganti F dengan id)
$E'T'id	id + id * id $	F → id	*top of stack = left most input : drop them !*
$E'T'	+ id * id $		M(T', +) = T'→ ε (ganti T' dengan ε)
$E'	+ id * id $	T'→ ε	M(E', +) = E'→ +TE' (ganti E' dg. TE')
$E'T+	+ id * id $	E'→ +TE'	*top of stack = left most input : drop them !*
$E'T	id * id $		M(T, id) = T → FT' (ganti T dengan FT')
$E'T'F	id * id $	T → FT'	M(F, id) = F → id (ganti F dengan id)
$E'T'id	id * id $	F → id	*top of stack = left most input : drop them !*
$E'T'	* id $		M(T', *) = T'→ *FT' (ganti T' dg. *FT')
$E'T'F*	* id $	T'→ ε	*top of stack = left most input : drop them !*
$E'T'F	id $		M(F, id) = F → id (ganti F dengan id)
$E'T'id	id $	F → id	*top of stack = left most input : drop them !*
$E'T'	$		M(T', $) = T'→ ε (ganti T' dengan ε)
$E'	$	T'→ ε	M(E', $) = E'→ ε (ganti E' dengan ε)
$	$	E'→ ε	*top of stack = left most input = $: finish !!* rangkaian token : id + id * id sesuai sintaks

LAPORAN PENDAHULUAN

1. Jelaskan tentang CFG !
2. Apa yang kamu ketahui tentang parsing ?

LAPORAN AKHIR

1. Jelaskan tugas dari analisa sintaks !
2. Jelaskan algoritma rekursif kiri !