# PERTEMUAN 9

# PENGURAIAN ATAS BAWAH (TOP DOWN PARSING)

# ANALISIS SINTAKS (1)

## A. TUJUAN PEMBELAJARAN

Setelah pembelajaran materi untuk pertemuan 9, mahasiswa mampu menggunakan prinsip dasar kerja penguraian dari atas ke bawah, dari metode yang paling umum serta berlanjut ke metode-metode lainnya.

## B. URAIAN MATERI

### 1. Pengertian Analisis Sintaksis

Analisis sintaksis adalah tahap analisis kedua di dalam tahapan analisis, yaitu suatu tahap untuk menyusun dan mengelompok mengelompokkan token token yang diperoleh dari tahap analisis leksikal ke dalam suatu hirarki tertentu yang secara keseluruhan mempunyai arti tertentu. Token-token tersebut dikirimkan dalam suatu bentuk pasangan terurut simbol, di mana komponen pertama dari pasangan terurut berisi lokasi atau alamat token di dalam Tabel simbol, sedangkan komponen keduanya adalah nomor representasi dari token yang bersangkutan.

Proses analisis sintaksis jauh lebih kompleks dari proses analisis leksikal. Analisis sintaksis ini akan menentukan struktur dari program secara keseluruhan. Proses analisis ini mirip dengan cara menentukan struktur suatu kalimat dalam suatu bahasa tertentu. Jadi harus ditentukan bagian mana yang merupakan " subjek", bagian mana yang merupakan "predikat", dan lain sebagainya. Dalam bahasa pemrograman pada umumnya, yang harus ditentukan adalah bagian mana yang menyatakan ekspresi matematika, bagian mana yang menyatakan suatu perintah, suatu prosedur dan lain sebagainya.

Hal yang dihasilkan oleh proses analisis sintaksis atau dikenal juga dengan nama proses penguraian adalah suatu pohon sintaks dimana daun dari pohon tersebut atau disebut dengan istilah node ujung adalah suatu token, sedangkan node yang lain menyatakan salah satu tipe dari kelompok sintaks

seperti ekspresi, perintah, prosedur dan lain-lain. Dalam hal ini pendekatan pohon sintaks yang digunakan mengacu pada sejumlah aturan tatabahasa yang digunakan untuk mendefinisikan secara tepat bahasa sumber. Jadi tatabahasa dapat digunakan oleh penganalisis sintaks untuk menentukan struktur dari program sumber.

Karena proses pengenalan struktur program sumber ini dikenal juga dengan proses penguraian, maka pohon sintaks yang diperoleh kadang-kadang disebut juga sebagai pohon urai dan penganalisis sintaks disebut juga pengurai atau parser.

Terdapat tiga metode umum pengurai untuk suatu tatabahasa, yaitu:

a. Metode-metode pengurai umum, seperti algoritma Cocke-Younger-Kasami dan algoritma earley's yang dapat mengurai semua jenis tatabahasa. Namun karena metode-metode ini sangat tidak efisien digunakan dalam pembuatan suatu kompilator, maka metode-metode ini tidak akan dibahas.
b. Metode atas bawah atau top-down yang membentuk pengurai pohon urai dari atas (akar) ke bawah (daun).
c. Metode bawah atas atau batam up yang membentuk pohon urai dari bawah daun dan bergerak ke akarnya atau keatas.

## 2. Tatabahasa Bebas Konteks

Tatabahasa bebas konteks atau (context-free grammar) adalah suatu tatabahasa yang tidak terikat pada arti dari bahasa yang tersusun tatabahasa bebas konteks ini dipilih karena terdapat banyak kegunaan, yaitu:

a. Untuk memudahkan pembentukan sintaks dari bahasa yang bersangkutan, walaupun di sisi lain penentuan semantic dari bahasa tersebut menjadi lebih sulit daripada sintaksnya.
b. Membantu penerjemahan suatu program.

Suatu tatabahasa bebas konteks secara ilmiah menerangkan struktur hierarki dari banyak bentuk bahasa pemrograman. Misalnya perintah *if-then-else* dalam pascal mempunyai bentuk umum perintah:

If (ekspresi) then perintah else perintah;

Dalam hal ini suatu perintah if-dan-else adalah suatu rangkaian karakter yang terdiri dari kata kunci if diikuti dengan simbol (ekspresi),perintah, kata kunci else, perintah, dan diakhiri dengan simbol;. Apabila digunakan nama variable ekspr untuk menyatakan suatu ekspresi, dan nama variable perintah untuk menyatakan perintah, maka struktur dari aturan ini dapat dinyatakan sebagai berikut:

Perintah → if ekspr then perintah else perintah;

Dimana tanda panah atau → dapat dibaca sebagai “dapat mempunyai bentuk”. Aturan seperti ini disebut dengan produksi. Dalam suatu produksi seperti ini, unsur-unsur leksikal seperti kata kunci *if, then, else* dan simbol () disebut dengan token-token. Sedangkan variable seperti ekspr dan perintah yang menyatakan barisan token-token disebut dengan non-terminal.

Secara lengkap, suatu tatabahasa bebas konteks mempunyai empat komponen, yaitu:

a. Himpunan token-token, yang disebut dengan sebagai simbol simbol terminal. Jika kita berbicara dalam bahasa program, kata token merupakan persamaan dari terminal.

b. Himpunan unsur-unsur non-terminal.

   Non-terminal adalah variable sintetik yang menyatakan himpunan dari string-string. Dalam contoh produksi di atas, perintah dan ekspr adalah suatu unsur non-terminal. Non-terminal mendefinisikan himpunan string-string yang membantu pendefinisian bahasa yang dibentuk berdasarkan tatabahasanya.

c. Himpunan produksi-produksi, di mana setiap produksi terdiri dari unsur non-terminal yang disebut bagian kiri dari produksi, anak panah, dan barisan token-token atau simbol-simbol non-terminal yang disebut bagian kanan dari produksi.

d. Salah satu dari unsur non-terminal ditentukan sebagai simbol awal, dan himpunan string-string yang dinyatakannya merupakan bahasa yang didefinisikan oleh tatabahasa yang bersangkutan.

Aturan umum yang digunakan untuk menentukan suatu tatabahasa adalah dengan menuliskan produksi-produksi yang ada dengan suatu produksi yang memuat simbol awal. Aturan lain adalah angka-angka, tanda-tanda seperti <=, dan *string-string* yang ditulis dengan huruf tebal seperti ***while*** disebut sebagai simbol terminal. Nama yang dicetak miring adalah simbol *non-terminal* dan nama atau simbol lain yang tidak dicetak miring adalah simbol *terminal*. Untuk memudahkan penulisan, produksi-produksi yang mempunyai simbol *non-terminal* pada bagian kiri sama, sedangkan bagian kanan dari produksi tidak sama, digunakan simbol | untuk memisahkan antar bagian kanan yang ada, dan simbol tersebut dibaca “atau”.

Contoh, apabila terdapat ekspresi yang terdiri dari angka-angka dan simbol positif dan negatif, yaitu 9-5 + 2, 3-1, dan 7. Perhatikan bahwa tanda positif dan negative harus timbul di antara dua angka, sehingga ekspresi-ekspresi seperti di atas dinyatakan sebagai “barisan angka-angka yang dipisahkan dengan tanda positif atau negatif”. Tatabahasa berikut ini adalah sintaks dari ekspresi-ekspresi yang dimaksudkan diatas.

Produksi yang ada adalah:

*Untai → untai + angka* *(1)*

*Untai → untai – angka* *(2)*

*Untai → angka* *(3)*

*Angka → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9* (4)

Bagian kanan dari tiga produksi dengan simbol non-terminal untai pada bagian kiri dapat dituliskan dalam bentuk lain yang ekuivalen, yaitu:

*Untai → untai + angka | untai – angka | angka*

Menurut definisi di atas maka token-token dari tatabahasa yang digunakan adalah simbol simbol:

+ - 0 1 2 3 4 5 6 7 8 9

Unsur-unsur non-terminal adalah yang dituliskan dengan cetak miring, yaitu *untai* dan *angka*, dengan *untai* sebagai simbol non-terminal awal, karena produksi produksinya adalah yang pertama diberikan.

Produksi untuk suatu unsur non-terminal adalah produksi yang mempunyai unsur non-terminal di bagian kiri produksi . String yang dibentuk dari token-token adalah suatu barisan dari nol atau lebih token. String yang mempunyai nol token, ditulis dengan simbol ε, yang disebut dengan *empty string*. Suatu tatabahasa diperoleh dari string string yang dimulai dengan simbol awal dan secara berulang-ulang mengganti unsur non-terminal dengan bagian kanan dari produksi non-terminal. String-string yang dapat diperoleh dari simbol awal membentuk suatu bahasa yang didefinisikan oleh tatabahasa yang bersangkutan.

Kembali pada contoh di atas, dari produksi:

*Untai → angka*

Maka sebuah angka yang berdiri sendiri adalah suatu untai juga. Sedangkan produksi:

*Untai → untai + angka*

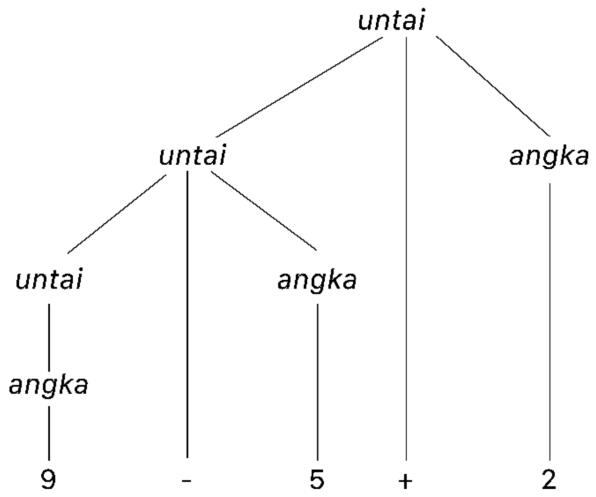
*Untai → untai – angka*

Menyatakan bahwa suatu untai yang diikuti dengan tanda positif atau negative dan sembarang angka akan membentuk suatu untai yang baru.

Ternyata bahwa semua produksi dalam keempat contoh produksi diatas diperlukan untuk mendefinisikan suatu bahasa. Sebagai contoh, suatu ekspresi 9-5+2 ada suatu untai, dengan perkataan lain 9-5+2 adalah anggota dari bahasa yang didefinisikan di atas. Hal ini ditunjukkan sebagai berikut:

a. Dari produksi (3), maka 9 adalah untai, karena 9 adalah suatu angka.
b. Dari produksi (2), maka 9 – 5 adalah suatu untai, dan 5 adalah angka.
c. Dari produksi (1), maka 9 – 5 + 2 adalah suatu untai, karena 9 - 5 adalah suatu untai dan 2 adalah suatu angka.

Hal tersebut di atas dapat digambarkan dengan sebuah pohon urai seperti pada gambar di bawah ini.




**Gambar 9.1** Pohon urai untuk ekspresi 9 – 5 + 2.

Pada gambar 9.1 di atas, setiap node pada pohon adalah label dari simbol pada tatabahasa tersebut. Node dalam dan anak-anaknya saling berhubungan dengan suatu produksi. Node dalam berhubungan dengan bagian kiri dari produksi, sedangkan anak-anaknya berhubungan dengan bagian kanan dari produksi.

Contoh yang lain, berikut ini adalah suatu tatabahasa dengan produksi produksinya mendefinisikan suatu ekspresi aritmatika yang sederhana:

*єksp → єksp op єksp*

*єksp → ( eksp )*

*eksp → - eksp*

*eksp → var*

*0p → +*

*0p → -*

*0p → **

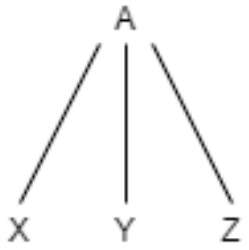
*0p → /*

*0p → ↑*

Dalam tatabahasa ini, simbol terminalnya adalah var + - * / ↑ ( ), dan simbol non-terminalnya adalah *eksp*, dan *op* dengan *eksp* sebagai simbol awal.

### 3. Pohon Urai

Pohon urai adalah suatu pohon yang menggambarkan bagaimana suatu simbol awal dari suatu tatabahasa menurunkan string dalam suatu bahasa. Bila suatu unsur non-terminal a mempunyai produksi A → XYZ maka pohon urai yang akan dibangun mempunyai suatu node dalam yang diberi label a yang mempunyai anak-anak yang diberi label X, Y, dan Z dengan urutan penempatan dari kiri ke kanan sebagai berikut:




**Gambar 9.2** Pohon urai untuk produksi A → XYZ

Secara formal, pohon urai dari suatu tatabahasa yang bersifat bebas konteks mempunyai sifat-sifat sebagai berikut:

a. Akar dari pohon urai diberi label simbol awal

b. Setiap daun dari pohon diberikan label token atau ε

c. Setiap node dalam diberi label non-terminal.

d. Jika a adalah unsur non-terminal yang merupakan label dari suatu node dalam, dan $X_1$, $X_2$, …, $X_n$ adalah label dari anak-anak node tersebut dari kiri ke kanan maka a → x1, x2, …, xn adalah suatu produksi. Dalam hal ini, setiap simbol dari x1, x2, …, xn dapat merupakan simbol dari unsur terminal maupun non-terminal . Dalam kasus khusus, yaitu a → ε maka node a mempunyai satu anak dengan label ε.

Pada contoh gambar 9.1., akar dari pohon urai diberi label untai, dan ini sebagai simbol awal. Anak-anak dari akar diberi label dari kiri ke kanan, seperti untai, +, dan angka, sesuai dengan produksi dari tatabahasa yang dibangun. Setiap pohon urai memberikan urutan dari kiri ke kanan pada daun-daunnya berdasarkan ide bahwa jika a dan b adalah dua anak dengan orang tua yang sama, dan a berada disebelah kiri b, maka semua turunan dari a akan terletak di sebelah kiri b.

**4. Ketentuan Notasi**

Untuk memudahkan penulisan dalam tatabahasa, yaitu untuk membedakan simbol mana yang menjadi simbol *terminal, non-terminal* dan sebagainya, maka berikut ini adalah ketentuan penulisan yang akan penulis pakai dalam pembahasan mengenai tatabahasa.

a. Simbol-simbol berikut ini adalah terminal:
   1) Huruf-huruf kecil awal abjad seperti a, b, c.
   2) Simbol-simbol operator seperti +, -.
   3) Simbol-simbol tanda baca seperti tanda kurung, koma.
   4) Angka-angka, yaitu 0,1 …, 9.
   5) String yang ditulis dengan huruf tebal, seperti **var, if**.

b. Simbol-simbol berikut ini adalah non-terminal:
   1) Huruf huruf besar awal abjad seperti a besar, b besar, c.
   2) Huruf besar s, apabila muncul dianggap sebagai simbol awal.
   3) Nama-nama dalam huruf kecil yang ditulis dengan huruf miring seperti ekspr, perintah.

c. Huruf-huruf besar akhir abjad seperti X, Y, Z mewakili simbol tatabahasa yang dapat berarti terminal atau non-terminal.

d. Huruf huruf kecil akhir abjad seperti u v w…, z mewakili string string dari suatu terminal.

e. Huruf-huruf kecil yunani seperti α, β, γ, mewakili string string dari simbol-simbol tatabahasa.

f. Jika A→ α1, A→ α2, …., a → $\alpha_x$ adalah semua produksi dengan satu non-terminal A di sebelah kiri; yang dapat disebut sebagai a-produksi; maka A-produksi tersebut dengan A→ $\alpha_1$, | $\alpha_2$ | … | $\alpha_1$, $\alpha_2$, $\alpha_3$, …., $\alpha_x$ disebut sebagai alternatif untuk A.

g. Apabila tidak disebutkan simbol mana yang menjadi awal produksi, maka sisi kiri dari produksi yang pertama kali dituliskan disebut sebagai simbol awal.

Contoh, dengan memakai ketentuan notasi di atas, maka tatabahasa pada contoh diatas sub bab ini dapat ditulis ulang sebagai berikut :

*E* → *E A E* | *(E)* | *-E* | var

*A* → + | - | * | / | ↑

dengan *E* dan *A* sebagai simbol non-terminal dan E simbol non-terminal awal.

## 5. Penguraian dari atas ke bawah (Top Down Parsing)

Penguraian dari atas ke bawah adalah sebuah metode penguraian yang menggunakan pola kerja pembentukan node dari pohon urai yang dimulai dari akar pohon dan berlanjut kea rah daun-daun dari pohon. Pada pembahasan mengenai metode penguraian dari atas ke bawah ini akan dibahas metode metode penguraian yang menggunakan prinsip dasar kerja penguraian dari atas ke bawah, dan akan dimulai dari metode yang paling umum serta berlanjut ke metode-metode lainnya yang merupakan kasus-kasus khusus dari metode umum tersebut.

a. Penguraian Turun Berulang (Recursive Descent Parsing)

Penguraian turun berulang adalah suatu metode penguraian dari atas ke bawah yang paling umum yaitu suatu usaha untuk mencari penurunan paling kiri dari suatu string masukan . Dapat dikatakan juga sebagai suatu usaha untuk membentuk pohon urai dengan masukkan dimulai dari akar pohon dan kemudian membentuk node-node dari pohon urai secara pre-order.

Untuk membahas penguraian turun berulang, berikut diberikan dua contoh tatabahasa yang cocok untuk dibahas dalam metode penguraian ini contoh 9.3 diberikan suatu tatabahasa yang dipakai untuk membentuk sebagian tipe dalam bahasa pemrograman pascal yaitu sebagai berikut:

*Type* → *sederhana*

|↑ **var**

| **array** [ *sederhana* ] *of type*

*Sederhana* → *integer*

| *char*

| **bilangan titik – titik bilangan**

Tatabahasa 2.

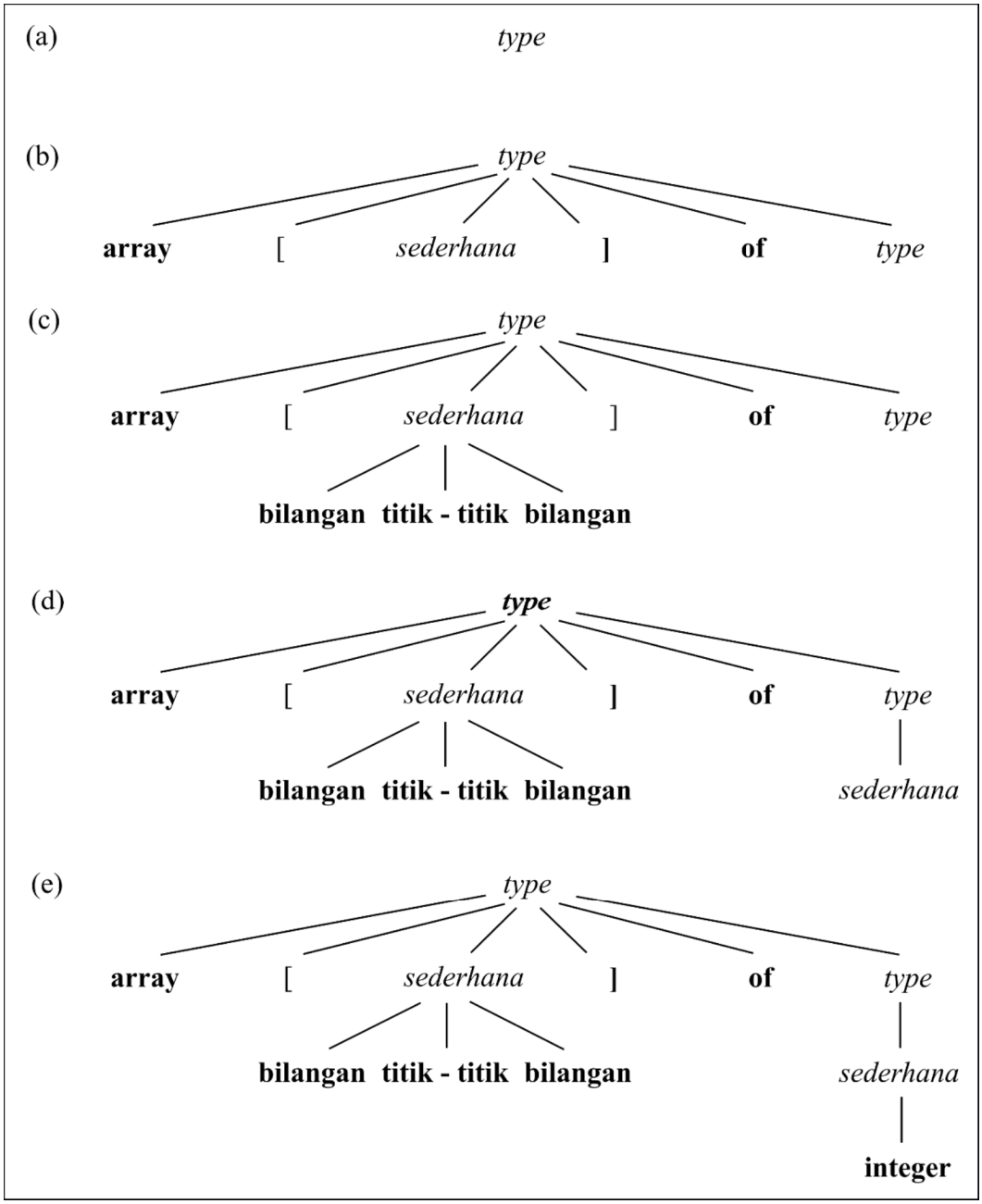
Pembentukan pohon urai dengan metode dari atas ke bawah dimulai dari akar pohon yang diberi label unsur non-terminal, dan secara berulang-ulang membentuk pohon urai dengan mengikuti 2 langkah berikut ini:

1) Pada node n dengan label non-terminal A, buatlah anak-anak dari n dengan label simbol-simbol dari sebelah kanan produksi A yang dipilih.
2) Cari node berikutnya di mana sub pohon urai akan dibentuk.

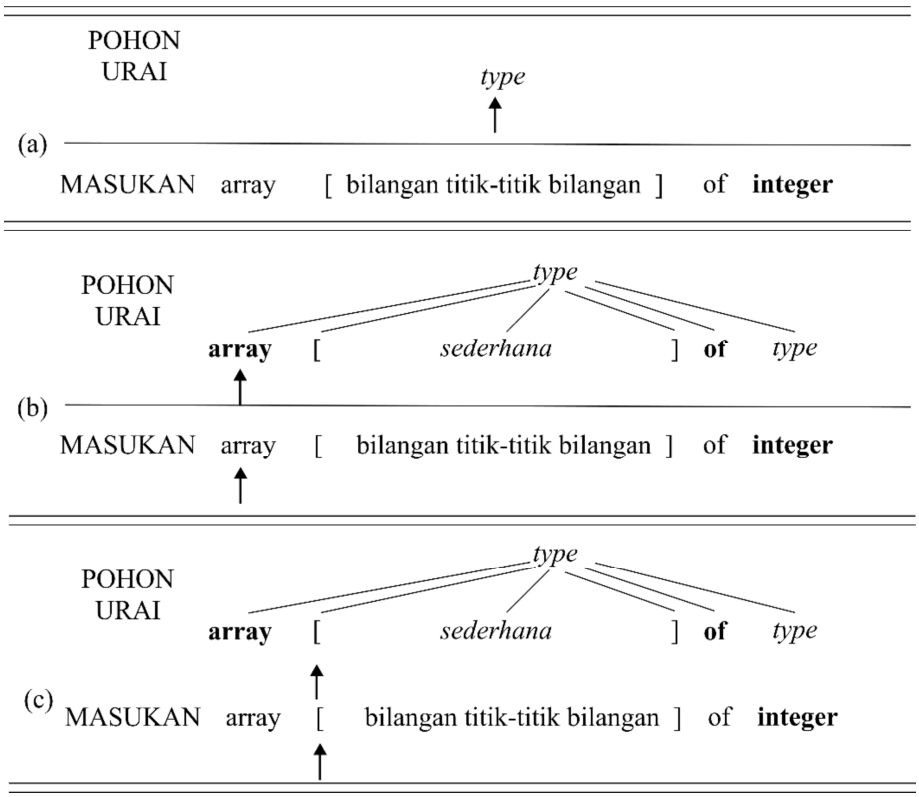
Gambar 9.3 memperlihatkan contoh hasil dari tahap-tahap pembentukan pohon urai dengan metode dari atas ke bawah untuk *string* **"array [ bilangan titik – titik bilangan ] of integer".**

Untuk beberapa tatabahasa langkah-langkah di atas dapat diimplementasikan selama pembacaan string masukan dari kiri ke kanan. Token yang sedang dibaca pada masukan string disebut dengan simbol lihat ke depan satu langkah atau lookahead. pada awalnya simbol lihat ke depan satu langkah adalah token pertama, yaitu yang paling kiri dari string masukkan. Gambar 9.4 di bawah menggambarkan proses penguraian *string* **"array [ bilangan titik – titik bilangan ] of integer".**

Mula-mula token array sebagai simbol lihat ke depan satu langkah dan merupakan bagian yang telah diketahui dari pohon urai yang terdiri dari akar yang diberi label non-terminal awal tipe seperti terlihat pada gambar 9.4(a). Tujuan yang akan dicapai adalah membentuk sisa pohon urai sehingga string yang dibentuk oleh pohon urai cocok dengan string masukan.




**Gambar 9.3** Tahap-tahap di dalam pembentukan dari atas-ke-bawah dari pohon urai.




**Gambar 9.4** Penguraian dari atas-ke-bawah bersamaan dengan pembacaan masukan dari kiri ke kanan.

Supaya terjadi kecocokan, non-terminal tipe seperti terlihat dalam gambar 9.4.(a) harus menurunkan string yang dimulai dengan simbol lihat ke depan satu langkah "array". Karena dari tatabahasa yang dipakai hanya ada satu produksi yang dapat membentuk string yang dimaksud maka produksi itulah yang dipilih untuk membentuk anak-anak dari akar yang diberi label simbol-simbol yang berada di sebelah kanan produksi yang dipilih.

Setiap bagian dari ketiga gambar yang terdapat dalam gambar 9.4. Mempunyai tanda panah yang menandai simbol lihat ke depan satu langkah pada masukan dan node dari pohon urai yang sedang diperhatikan. Ketika anak-anak dari node telah dibentuk, maka yang harus diperhatikan adalah anak yang paling kiri. Pada gambar 9.4(b) anak-anak dari akar pada pohon urai telah terbentuk, dan anak-anak paling kiri dengan label array sedang diperhatikan.

Ketika node yang sedang diperhatikan pada pohon urai adalah suatu simbol terminal dan simbol terminal tersebut cocok dengan simbol lihat ke depan satu langkah, maka proses pada pohon urai dan masukan dapat dilanjutkan. Token berikutnya dari masukan kemudian menjadi simbol lihat ke depan satu langkah yang baru dan yang diperhatikan adalah anak berikutnya dari pohon urai . Pada gambar 9.4.(c) anak panah di dalam pohon urai telah dimajukan pada token berikutnya yaitu token [. Setelah dilanjutkan lagi anak panah dalam pohon urai akan menunjukkan pada anak yang berlabel dengan non-terminal sederhana. Ketika node dengan label non-terminal sedang diperhatikan, dalam contoh ini noda dengan label sederhana, kita kemudian mengulang lagi proses untuk memilih produksi dari non-terminal ini.

Pada umumnya pemilihan produksi non-terminal bersifat coba-coba, artinya produksi yang kita pilih mungkin tidak cocok sehingga harus dilakukan proses mundur kembali atau backtracking untuk mencoba produksi yang lain. Suatu produksi dikatakan tidak cocok apabila setelah menggunakan produksi tersebut tidak dapat diperoleh pohon urai yang cocok dengan string masukkan.

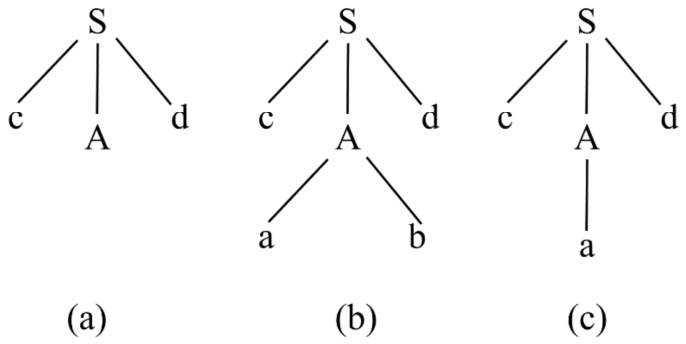
Contoh 9.4 Andaikan diketahui suatu tatabahasa sebagai berikut:

$$S \rightarrow cad$$

$$A \rightarrow ab \mid a$$

(Tatabahasa 2)

Dengan string masukan w = cad. Untuk membentuk pohon urai untuk string w tersebut dari atas ke bawah, pertama kali buat suatu pohon yang mempunyai sebuah node tunggal yang diberi label S. Pointer masukkan menunjuk ke c sebagai simbol pertama dari w. gunakan produksi pertama dari S, untuk mengembangkan pohon urai, sehingga didapat pohon urai seperti pada gambar 9.5.(a).




**Gambar 9.5** tahap-tahap dalam penguraian atas ke bawah

Daun yang paling kiri dari gambar 9.5.(a), dengan label c, bersesuaian dengan simbol pertama dari w, sehingga pointer dapat dimajukan ke a, sebagai simbol kedua dari w. Langkah selanjutnya adalah memperhatikan daun kedua dengan label A. Gunakan produksi A pada pilihan pertama, yaitu produksi A → ab, sehingga diperoleh pohon urai seperti gambar 9.5.(b). Daun paling kiri dengan label a ternyata cocok dengan simbol kedua dari w, sehingga pointer dapat dimajukan ke d, yaitu simbol ketiga dari w. langkah berikutnya bandingkan d dengan daun berikutnya. ternyata daun berikutnya berlabel b, sehingga tidak cocok, maka laporkan bahwa ada kegagalan penguraian. Selain itu kembali ke A lagi untuk melihat apakah ada alternatif dari A yang lain yang belum dipakai dan mungkin lebih cocok.

Pada saat kembali ke A, pointer masukkan dikembalikan ke posisi 2, yaitu posisi ketika pertama kali mengembangkan A, dan ini berarti prosedur untuk A harus memasukkan pointer ke dalam variabel lokal. Langkah berikutnya adalah mencoba alternatif kedua dari A yaitu produksi A → a, sehingga diperoleh gambar 9.5.(c). Daun dengan label a yang diperoleh ternyata cocok dengan simbol kedua w, sehingga seperti diterangkan di atas, pointer dapat dimajukan dan ternyata ada dengan label d juga cocok dengan simbol ketiga dari w sampai disini proses pembentukan pohon urai untuk w selesai, dan penguraian secara komplit sudah berhasil.

Tatabahasa pengulangan kiri dapat menyebabkan penguraian turun-berulang, walaupun dengan pelacakan mundur, menjadi suatu prosedur loop yang tidak hingga. Dari contoh 9.4, hal ini mungkin terjadi pada saat mencoba untuk mengembangkan A lagi tanpa adanya masukkan.

Sekalipun demikian, ada suatu hal khusus, yaitu penguraian turun berulang yang tidak menggunakan backtracking yang disebut dengan penguraian prakira (predictive parsing) seperti akan dibahas dalam pembahasan berikut ini.

b. Penguraian Prakira (Predictive Parsing)

Penguraian prakira seperti telah disebutkan dalam pembahasan sebelumnya adalah suatu metode penguraian turun berulang yang tidak memerlukan adanya backtracking. Jadi apabila terdapat simbol masukan a dan suatu non terminal A dimana adalah A suatu produksi yang berbentuk $A \rightarrow a_1 \mid a_2 \mid \ldots \mid a_n$., maka pengurai harus mengetahui dengan pasti alternatif mana yang harus dipilih dari produksi A sehingga dihasilkan suatu string yang dimulai dengan a. Dengan perkataan lain harus dapat mendeteksi produksi yang tepat dipakai untuk suatu masukan dengan adanya melihat dari simbol yang pertama kali diturunkan.

Contoh 9.5. Diberikan produksi sebagai berikut:

*Pernyataan* → **if** *ekspr* **then** *pernyataan* **else** *ekspr*

| **while** *ekspr* **do** *pernyataan*

| **begin** *untai_pernyataan* **end**

Dari produksi tersebut kata kunci **if, while** dan **begin** sudah memberikan informasi tentang alternatif produksi mana yang mungkin berhasil untuk dipakai apabila terdapat suatu perintah masukan.

1) Diagram Transisi Untuk Pengurai Prakira

Diagram transisi untuk penganalisis sintaks atau pengurai berbeda dengan diagram transisi untuk menganalisis leksikal. Dalam diagram transisi untuk pengurai, hanya akan terdapat satu diagram untuk setiap simbol non-terminal. Label-label dari garis-garis dapat berupa token token atau non terminal non terminal. Apabila terdapat suatu transisi pada token atau terminal berarti token itu merupakan simbol masukkan berikutnya, namun apabila terdapat transisi pada non terminal A berarti berikutnya adalah proses pemanggilan prosedur untuk A.

Untuk membuat diagram transisi suatu pengurai prakira dari suatu tatabahasa, langkah pertama adalah dengan menghilangkan rekursif kiri dari tatabahasa tersebut, dilanjutkan dengan melakukan faktorisasi kiri dari hasil tatabahasa yang telah dieliminasi. kemudian pada setiap non terminal A, lakukan langkah-langkah berikut ini:

a) Buat state awal dari state akhir.

b) untuk setiap produksi A → $X_1$ $X_2$ ... $X_n$ buat suatu jalur dari state awal ke state akhir, dengan garis-garis yang berlabel $X_1$ $X_2$ ... $X_n$

Cara kerja pengurai prakira dalam diagram transisi adalah sebagai berikut

Dimulai dari state awal untuk simbol awal

a) Jika setelah beberapa langkah dan proses berada di state s dengan garis yang berlabel terminal a ke state t, dan jika masukkan berikutnya adalah simbol a, maka pengurai memindahkan kursor masukan satu posisi ke kanan dan menuju ke statet.

b) Di lain pihak, jika garis tadi berlabel suatu non terminal A, maka pengurai akan menuju ke state awal untuk A tanpa memindahkan kursor masukkan. Jika telah mencapai state akhir dari A, maka pengurai langsung menuju ke state t, sebagai akibat membaca A dari masukkan pada saat terjadi perpindahan dari state s ke statet.

c) Akhirnya, jika garis dari s ke t berlabel ε, maka dari state s pengurai akan langsung menuju ke state t tanpa memajukan masukan.

Contoh 9.5. diberikan tatabahasa sebagai berikut:

E → TE'

E → +TE' | ε

T → FT'

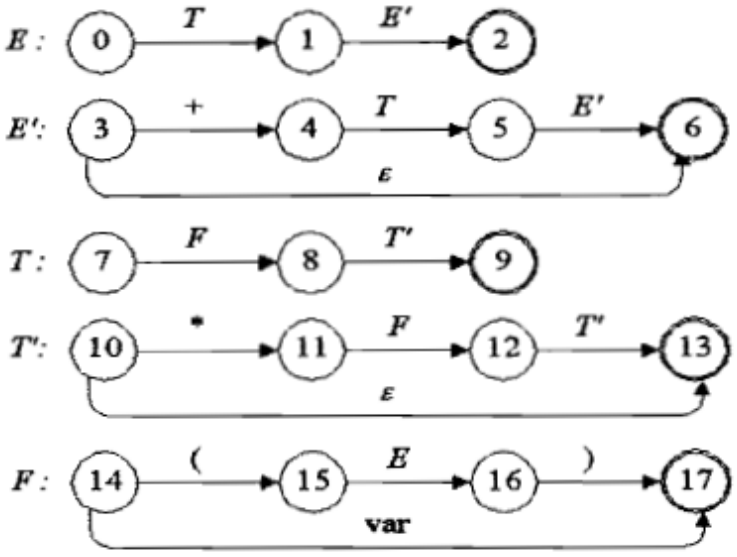
T' → *FT' | ε

F → (E) | **var**

(Tatabahasa 3)

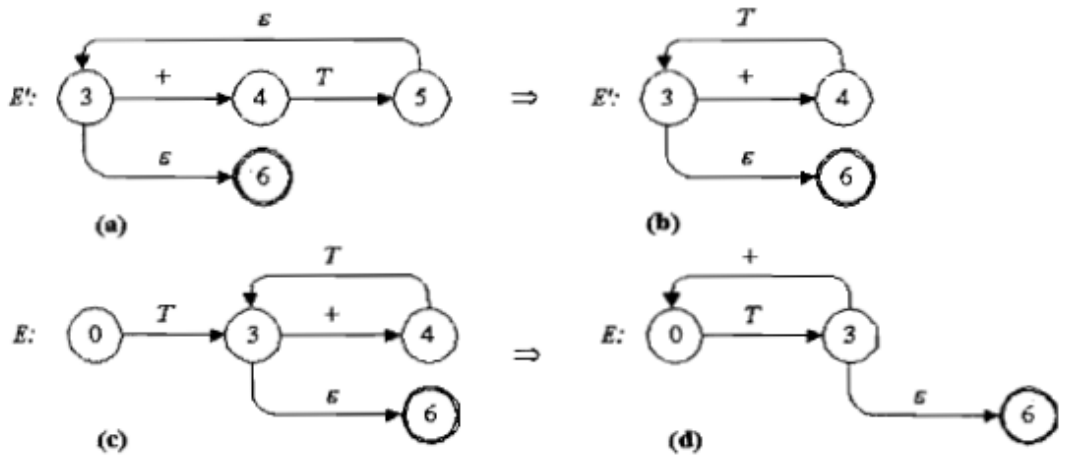
Maka kumpulan diagram transisi untuk tatabahasa tersebut seperti terlihat dalam gambar 9.6 di bawah.

Dalam tatabahasa ini terdapat ambigu, yaitu apakah kita akan memakai garis ε atau tidak, seperti terlihat pada diagram transisi untuk E' dan T'. jika kita interprestasi garis yang keluar dari state awal E', maka jika + merupakan masukan yang berikutnya berarti pilihan yang diambil adalah transisi pada +, tetapi jika bukan, maka ambil transisi pada ε. Hal ini analog untuk diagram pada T', maka ambiguiti akan hilang, dan kita dapat menulis program pengurai prakira untuk tatabahasa (3) tersebut.




**Gambar 9.6** Diagram transisi untuk tatabahasa (3).

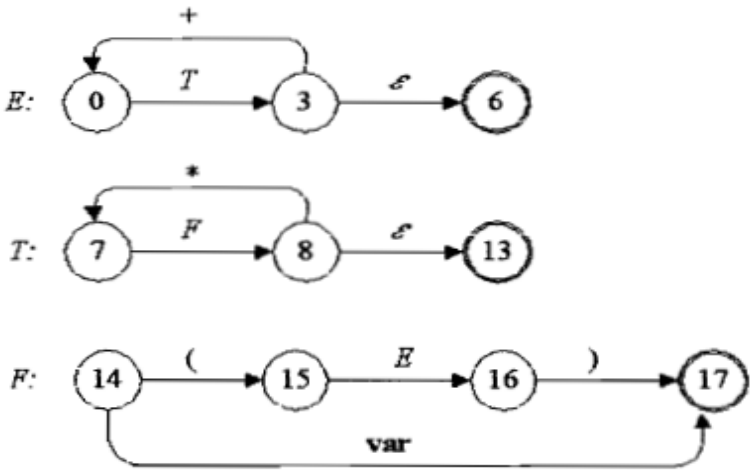
Diagram diagram transisi dapat disederhanakan dengan melakukan substitusi diagram-diagram satu dengan yang lain. Sebagai contoh, dalam gambar 9.7 (a) pemanggilan e pada dirinya sendiri dapat diganti dengan meloncat ke awal dari diagram untuk e. Diagram ini dapat disederhanakan lagi seperti pada gambar 9.7 (b).




**Gambar 9.7** Penyederhanaan dari diagram transisi pada gambar 9.6

Penyederhanaan masih dapat dilanjutkan yaitu dengan mensubstitusi diagram pada gambar 9.7.(b) untuk transisi pada *E'* pada diagram untuk *E* dari gambar 9.6., sehingga akan dihasilkan diagram transisi seperti terlihat pada gambar 9.7 (c). Dari gambar 9.7 (c) ini kita dapat melihat bahwa node pertama dan ketiga ekuivalen, sehingga kita dapat menyatukan kedua node tersebut. Hasil akhir dari penyederhanaan dapat dilihat dalam gambar 9.7 (d).

Teknik yang sama dapat kita terapkan pada diagram untuk *T* dan *T'* sehingga kita memperoleh gambar diagram lengkap untuk tatabahasa (3) seperti terlihat dalam gambar 9.8 berikut ini.




**Gambar 9.8** diagram - diagram transisi yang telah disederhanakan untuk ekspresi aritmatika.

2) FIRST dan FOLLOW

Pembentukan pengurai prakira dibantu oleh dua buah fungsi yang berhubungan dengan sebuah tatabahasa G. Fungsi tersebut adalah fungsi first dan follow yang akan memperbolehkan dilakukannya pengisian kolom masukan tabel penguraian prakira dari tatabahasa G tersebut.

Jika α adalah sebarang string dari suatu simbol tatabahasa, maka first (α) adalah suatu himpunan terminal-terminal yang menjadi awal dari string-string yang diturunkan dari α. Jika α => ε, maka ε€FIRST(α). berikut ini adalah aturan yang dipakai untuk mencari FIRST(X) untuk semua simbol tatabahasa X.

Aturan mencari FIRST(X) untuk semua simbol tatabahasa X ini dipakai sampai tidak ada lagi terminal atau ε yang dapat ditambahkan kedalam himpunan dari hasil FIRST(X) tersebut.

a) Jika X adalah suatu terminal, maka FIRST(X) adalah {X}.

b) Jika $X \rightarrow \varepsilon$ adalah suatu produksi, maka ε€ FIRST(X).

c) jika x adalah suatu non-terminal $X \rightarrow Y_1 Y_2 \ldots Y_K$ adalah suatu produksi dan jika untuk setiap i sedemikian sehingga setiap $Y_1 Y_2 \ldots Y_{i-1}$ adalah non-terminal – non-terminal dan FIRST($Y_j$) untuk semua j = 1, 2, ..., i-1 atau dengan perkataan lain $Y_1 \ldots Y_{i-1}$ maka setiap terminal dalam FIRST($Y_j$) dimasukkan ke dalam FIRST(X). jika ε didalam FIRST($Y_j$) untuk semua j = 1, 2 ,.., k, maka tambahkan ε ke dalam FIRST(X).

untuk membuat garis sebarang $X_1 X_2 \ldots X_n$ caranya adalah:

a) Tambahkan ke dalam FIRST ($X_1 X_2 \ldots X_n$) semua simbol – simbol bukan ε dari FIRST($X_1$).

b) Tambahkan juga simbol – simbol non-ε dari FIRST($X_2$) εƐ FIRST($X_1$), simbol – simbol non-ε dari FIRST($X_3$) jika εƐ FIRST($X_2$), dan seterusnya.

c) Langkah terakhir, tambahkan ε ke dalam FIRST($X_1 X_2 \ldots X_n$) jika ε FIRST($X_1$), untuk semua i.

Contoh 9.4 Dari tatabahasa (3) terdapat produksi-produksi sebagai berikut:

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \varepsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow * FT' \mid \varepsilon$$

$$F \rightarrow (E) \mid \boldsymbol{var}$$

(Tatabahasa 3)

Maka dengan mempergunakan amaka dengan mempergunakan aturan-aturan di atas, maka dapat dicari FIRST dari setiap non-terminal dalam tata Bahasa tersebut, yaitu sebagai berikut:

a) Dengan menggunakan aturan nomor 3). untuk FIRST dengan i=1 pada masing-masing alternatif, maka FIRST(F) = {(, var}, karena berdasarkan aturan nomor 1). FIRST('(') = {(}, dan FIRST (var) = {var}.

b) Dengan menggunakan aturan nomor 3). Untuk i=1, produksi T → FT', maka FIRST (T) = FIRST (F) = {(, var}.

c) Dengan menggunakan aturan nomor 3) untuk i=1, produksi E → TE', maka FIRST (T) = FIRST(F) = {(var}.

d) Dengan menggunakan aturan nomor 3). Untuk first dengan i = 1 pada masing-masing alternatif, maka FIRST(E') = {+, s} karena berdasarkan aturan nomor 1). First ('+') = {+}, dan berdasarkan aturan nomor 2). Pada produksi E'→ ε maka ε∈ FIRST(E').

e) Dengan menggunakan aturan nomor 3). Untuk first dengan i = 1 pada masing-masing alternatif, maka first (T') = (*, ε), karena berdasarkan aturan nomor 1). FIRST('*') = {*}, dan berdasarkan aturan nomor 2). pada produksi T' → ε maka ε∈ FIRST(T').

Dari Langkah 1). Sampai dengan 5). Diperoleh FIRST(E) = FIRST(T) = FIRST(F) FIRST(E') = {+, ε}, dan FISRT(T') = {*, ε}.

Penggunaan fungsi first dalam prosedur pengurai prakira

sangat jelas, yaitu untuk memandu pengurai prakira dalam memilih alternatif produksi mana yang sesuai sehingga proses penguraian akan berhasil. Persoalan terjadi pada saat ε berada di dalam suatu first. Berkaitan dengan inilah maka dimunculkan fungsi follow yang akan bekerja Bersama – sama dengan fungsi first untuk mencegah terjadinya suatu kesalahan pemilihan produksi pada saat ditemukan ε dalam suatu first.

FOLLOW(A), dengan A adalah suatu non-terminal, didefinisikan sebagai himpunan terminal α yang dapat muncul tepat disebelah kanan dari A dalam string dari suatu tatabahasa yang diturunkan dari simbol awal. Dengan perkataan lain suatu himpunan terminal α, jika S => αAaβ untuk suatu α dan β. Jika A merupakan simbol paling kanan, maka $ berada di dalam FOLLOW(A).

Untuk mencari FOLLOW(A) dengan suatu non-terminal, dipakai aturan berikut ini sampai tidak ada lagi yang dapat ditambahkan ke dalam himpunan FOLLOW:

a) Masukan $ ke dalam FOLLOW(A), bila A dari simbol awal. $ adalah simbol akhir dari suatu masukan.

b) Jika terdapat produksi A→αββ dan β≠ε, maka semua FIRST(β) kecuali ε dimasukkan ke dalam FOLLOW(B).

c) Jika terdapat produksi A→αβ atau produksi A→αBβ di mana FIRST(β) memuat ε, maka semua yang berada di dalam FOLLOW(A) dimasukkan ke dalam FOLLOW(B).

Contoh 9.5. Dengan mempergunakan tatabahasa (3), dan dengan memakai aturan-aturan di atas, maka dapat dicari FOLLOW(E), FOLLOW(T), dan follow untuk nonterminal lainnya, yaitu sebagai berikut:

a) Berdasarkan aturan nomor 1), maka $ƐFOLLOW(E). Berdasarkan aturan nomor 2) pada produksi F→(E), ‘)’ ∈ FOLLOW(E). Sehingga FOLLOW(E) = { ), $}.

b) Dengan menggunakan aturan nomor 3). Pada produksi E→TE’ dimana ε∈ FIRST(E’), maka FOLLOW(E’) = FOLLOW(E) = { ), $}.

c) Dengan dengan menggunakan aturan nomor 2). Pada produksi E→TE', maka semua yang terdapat di dalam FIRST(E') Kecuali ε adalah anggota dari FOLLOW(T), sehingga +∈ FOLLOW(T).

Dengan menggunakan aturan nomor 3). pada produksi E→TE' di mana ε∈*FIRST(E')*, maka semua anggota FOLLOW(E) adalah anggota follow(T). Sehingga FOLLOW(T) = { +, ), $}.

d) Dengan menggunakan aturan nomor 3). pada produksi T→FT' dimana ε∈FIRST(T'), maka semua anggota follow(T) adalah anggota FOLLOW(T'), sehingga FOLLOW(T') = { +), $}.

e) Dengan menggunakan aturan nomor 2). Pada produksi T→FT', maka semua yang terdapat di dalam FIRST(T') kecuali ε adalah anggota dari FOLLOW(F), sehingga *∈FOLLOW(T). Dengan menggunakan aturan nomor 3). Pada produksi T→FT' di mana ε∈FIRST(T'), maka semua anggota FOLLOW(T) adalah anggota FOLLOW(F). sehingga FOLLOW(F) = {*,+,), $}.

Jadi FOLLOW(E) = FOLLOW(E') = {), $}, FOLLOW(T) = FOLLOW(T') = {+,), $}, dan FOLLOW(X) = {*, +,), $}.

## C. SOAL LATIHAN/ TUGAS

1. Kenapa dilakukan penghilangan Left Recursive dan Left Factory pada Top-down parsing?
2. Apa perbedaan Top-down dan Bottom-up?serta mana yang lebih bagus dan buktikan!
3. Dalam syntax analyzer, buatlah 1 case untuk melakukan proses syntax analyzer! (Menggunakan Teori J.P Bennet)
4. Apa yang dimaksud dengan Top Down Parsing? Jelaskan!
5. Sebutkan metode – metode yang ada pada Top Down Parsing!

## D. REFERENSI

Aho,A.V.,R.Sethi, and J. D. Ullmann, 1988. *Compiler*: Principles, Techniques, and Tools. Massachusetts: Addison Wesley Publishing Company.

Tremblay, Jean-Paull, Paul G. Sorenson, 1985. The Teory and Practice of *Compiler*, New York : McGraw-Hill Co. (Buku Pegangan Pendamping)

Sukamdi, Merekayasa *Interpreter*. 1995. ( Sebuah Penerapan Teknik Kompilasi), Jakarta: PT Elex Media Komputindo. (Buku Pegangan Pendamping)

Firrar Utdiartomo, 2001. Teknik Kompilasi, Yogyakarta: J&J Learning. (Buku Pegangan Pendamping)

Sumantri Slamet, Heru,S. 1995. Teknik Kompilasi, Jakarta: PT. Elex Media Komputindo.

# GLOSARIUM

**Integer** adalah sebuah tipe datayang mempresentasikan bilangan bulat.

**Ambigu** adalah kemampuan mengekspresikan lebih dari satu penafsiran.

**Backtracking** adalah suatu algoritma umum yang dipakai untuk menemukan solusi dari permasalahan komputasi.

**Array** adalah tipe data terstruktur yang dapat menyimpan banyak data dari tipe data dan nama yang sama.