TEKNIK KOMPILASI

Pertemuan 4 :
TOP DOWN PARSING

PROGRAM STUDI : TEKNIK INFORMATIKA

UNIVERSITAS INDRAPRASTA PGRI

JL. Nangka No. 58 C Tanjung Barat Jagakarsa, Jakarta Selatan

JL. Raya Tengah, Kelurahan Gedong, Pasar Rebo, Jakarta Timur

Parsing Di Golongkan Menjadi

- **Top Down**
- Penelusuran dari root ke leaf atau dari simbol awal ke simbol terminal
- metode ini meliputi:
- Backtrack/backup : Brute Force
- No backtrack : Recursive Descent Parser

Bottom-Up

- Metode ini melakukan penelusuran dari leaf ke root

Parsing: Brute force

- Memilih aturan produksi mulai dari kiri
- Meng-expand simbol non terminal sampai pada simbol terminal
- Bila terjadi kesalahan (string tidak sesuai) maka dilakukan backtrack
- Algoritma ini membuat pohon parsing secara top-down, yaitu dengan cara mencoba segala kemungkinan untuk setiap simbol non-terminal
- Contoh suatu language dengan aturan produksi sebagai berikut

  $S \rightarrow aAd \mid aB$

  $A \rightarrow b \mid c$

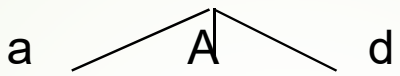
  $B \rightarrow ccd \mid ddc$

- Misal ingin dilakukan parsing untuk string ‘accd’

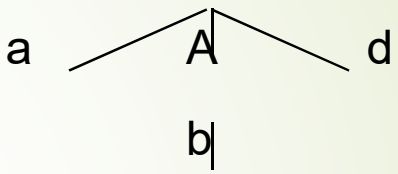
Parsing: Brute force

(i) S

(ii) S

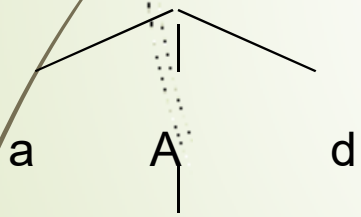


(iii) S

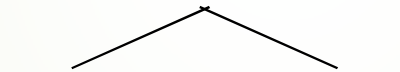


Terjadi kegagalan (iii), dilakukan back track

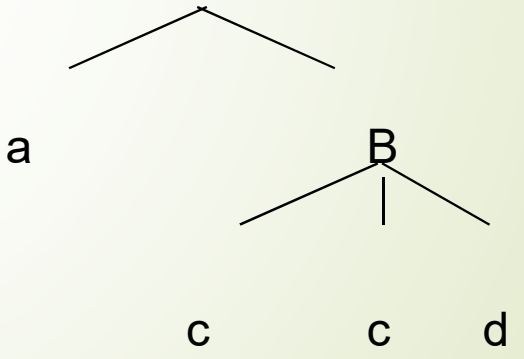
(iv) S



(v) S



(vi) S



Terjadi kegagalan lagi (iv), dilakukan back-track

Parsing: Brute force

Kelemahan dari metode-metode brute-force

- Mencoba untuk semua aturan produksi yang ada sehingga menjadi lambat (waktu eksekusi)
- Mengalami kesukaran untuk melakukan pembetulan kesalahan
- Memakan banyak memakan memori, dikarenakan membuat backup lokasi backtrack
- Grammar yang memiliki Rekursif Kiri tidak bisa diperiksa, sehingga harus diubah dulu sehingga tidak rekursif kiri, Karena rekursif kiri akan mengalami **Loop** yang terus-menerus

Brute Force : Aturan Produksi

Aturan Produksi yang rekursif memiliki ruas kanan (hasil produksi) yang memuat symbol variable pada ruas kiri

Sebuah produksi dalam bentuk :

$A \rightarrow \beta\ A$ *Merupakan produksi rekursif kanan*

$\beta =$ *berupa kumpulan simbol variabel dan terminal*

Contoh

S→dS

B →adB

Bentuk Produksi yang rekursif Kiri :

$A \rightarrow A\ \beta$ *Merupakan Produksi Rekursif Kiri*

Contoh :

$S \rightarrow$ Sd

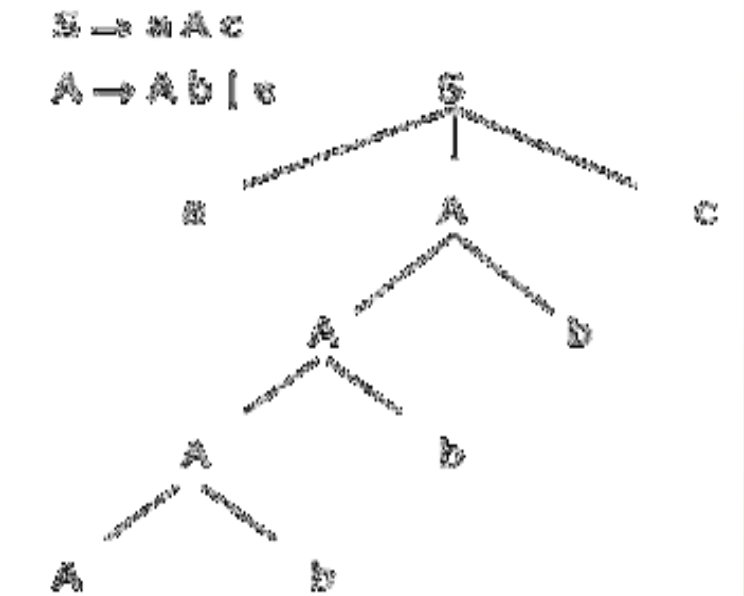
B → Bad

Aturan produksi : Brute force

Produksi yang rekursif kanan akan menyebabkan penurunan tumbuh kekanan,

Sedangkan produksi yang rekursif kiri akan menyebabkan penurunan tumbuh ke kiri.

Contoh: Context free Grammar dengan aturan produksi sebagai berikut:



Aturan produksi : Brute force

Dalam Banyak penerapan tata-bahasa, ***rekursif kiri*** tidak diinginkan, Untuk menghindari penurunan kiri yang looping, perlu dihilangkan sifat rekursif, dengan langkah-langkah sebagai berikut:

- Pisahkan Aturan produksi yang rekursif kiri dan yang tidak; misalnya

Aturan produksi yang **rekursif kiri**

$$A \rightarrow A\alpha 1 \mid A\alpha 2 \mid \ldots \mid A\alpha n$$

Aturan produksi yang ***tidak*** **rekursif kiri**

$$A \rightarrow \beta 1 \mid \beta 2 \mid \ldots \mid \beta n$$

Ganti aturan yang rekursif kiri dengan aturan berikut:

$$A \rightarrow \beta_1 Z \mid \beta_2 Z \mid \ldots \mid \beta_n Z$$
$$Z \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$$
$$Z \rightarrow \alpha_1 Z \mid \alpha_2 Z \mid \ldots \mid \alpha_n Z$$

Aturan produksi : Brute force

- Pergantian dilakukan untuk setiap aturan produksi dengan simbol ruas kiri yang sama, bisa muncul variabel Zf, Z2 dst, sesuai dengan variabel yang menghasilkan rekurisif kiri

  Contoh: Tata Bahasa Context free

  S → Sab | aSc | dd | ff | Sbd

- Pisahkan aturan produksi yang rekursif kiri

  S → Sab | Sbd

  Ruas Kiri untuk S: α=ab , $\alpha 2 = bd$

- Aturan Produksi yang tidak rekursif kiri

  S → aSc | dd | ff

  dari situ didapat untuk Ruas Kiri untuk S: $\beta 1$= aSc $\beta 2$= dd $\beta 3$= ff

Aturan produksi : Brute force

- Langkah berikutnya adalah penggantian yang rekursif kiri

S → Sab | Sbd, dapat digantikan dengan

S → aSc***Z1*** | dd***Z1*** | ff***Z1***

Z1 → ab | bd

Z1 → ab***Z1*** | bd***Z1***

- Hasil akhir yang didapat setelah menghilangkan rekursif kiri adalah sebagai Berikut:

S → aSc | dd | ff

S → aSc***Z1*** | dd***Z1*** | ff***Z1***

Z1 → ab | bd

Z1 → ab***Z1*** | bd***Z1***

Aturan produksi : Brute force

- Kalau pun tidak mungkin menghilangkan rekursif kiri dalam penyusunan aturan produksi maka produksi rekursif kiri diletakkan pada bagian belakang atau terkanan, hal ini untuk menghindari looping pada awal **proses parsing**

- Metode ini jarang digunakan, karena semua kemungkinan harus ditelusuri, sehingga butuh waktu yang cukup lama serta memerlukan memori yang besar untuk penyimpanan stack (backup lokasi backtrack)

- Metode ini digunakan untuk aturan produksi yang memiliki alternatif yang sedikit

Parsing: Recursive Descent Parser

Parsing dengan **Recursive Descent Parser**

- Salah satu cara untuk meng-aplikasikan bahasa context free
- Simbol terminal maupun simbol variabelnya sudah bukan sebuah karakter
- Besaran leksikal sebagai simbol terminalnya, besaran syntax sebagai simbol variablenya /non terminalnya
- Dengan cara penurunan secara recursif untuk semua variabel dari awal sampai ketemu terminal
- Tidak pernah mengambil token secara mumdur (back tracking)
- Beda dengan turing yang selalu maju dan mundur dalam melakukan parsing

RECURSIVE DESCENT PARSER

- Menampilkan aturan grammar untuk sebuah nonterminal A sebagai definisi dari sebuah procedure, yang dapat dikenal sebagai procedure A.
- Algoritma proceudure Recursive descent perser :

1. Sisi kanan dari produksi A merupakan struktur kode untuk procedure.
2. Urutan terminal dan nonterminal sesuai dengan input dan memanggil procedure lainnya,

- **Jika bertemu terminal maka lakukan pemilihan sesuai dengan kode (if dan case).**
- **Jika bertemu nonterminal maka panggil procedure dengan nama yang sama dengan nama nonterminal**

Aturan Produksi memakai Recursif Descent :

- Semua simbol variabel dijadikan prosedur/fungsi
- Jika ketemu simbol terminal pada aturan produksi , maka panggil prosedurnya
- Penelusuran bersifat top down mengikuti sintaks sesuai pola pada diagram sintaks
- Fungsi/prosedur ditulis untuk setiap non terminal dari suatu produksi. Setiap fungsi/prosedur akan melemparkan nilai benar atau salah bergantung pada apakah fungsi tersebut mengenali substring yang diterima sebagai ekspansi dari non terminal.

Contoh :
Diberikan sebuah grammar berikut :
Factor → (exp) | number

Maka Pseudocode untuk factor:

```
procedure factor ;
begin
  case token of
   ( :    match( ( ) ;
          exp ;
          match( ) ) ;
  number :
          match(number) ;
  else error ;
  end case ;
end factor ;
```

Dimana Pseudocode untuk prosedur match :

```
procedure match ( expectedToken ) ;
begin
  if token = expectedToken then
     getToken ;
  else
     error ;
  end if ;
end match ;
```

Contoh :

Grammar dengan BNF :

<program> ::= t_PROG t_ID t_SEMICOL <block> t_DOT

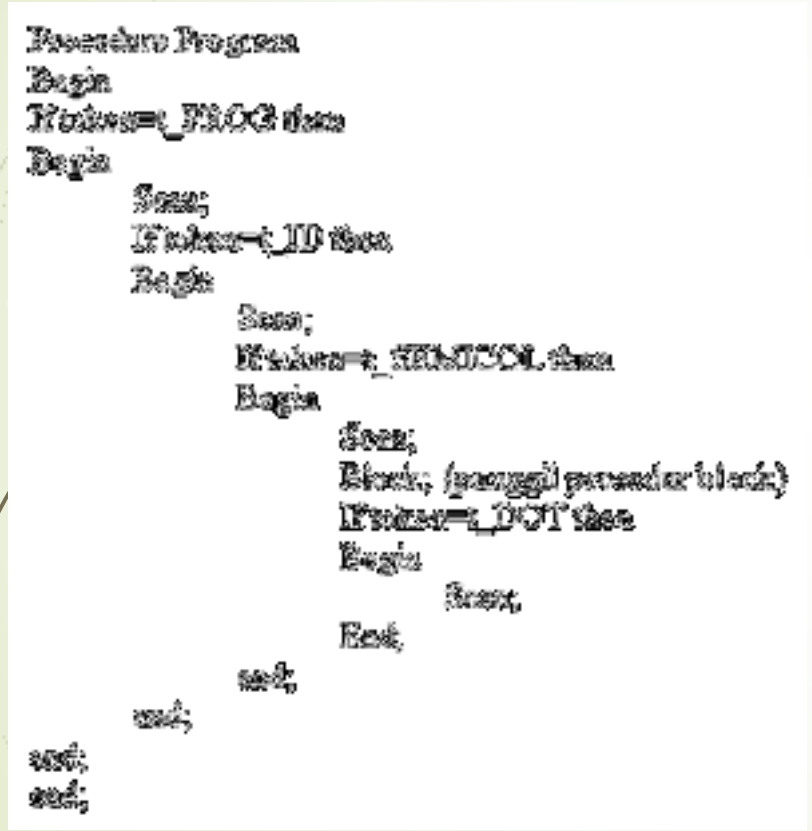
<block> ::= t_BEGIN <statement> {t_SEMICOL <statement>}
t_END

<statement> ::= t_ID t_ASS <simple exp> |
t_IF <exp> t_THEN <statement>|
t_IF <exp> t_THEN <statement> t_ELSE
<statement>

<exp> ::= <simple_exp> t_EQ <simple exp> |
<simple_exp> t_LT <simple_exp>|
<simple_exp> t_GT <simple_exp>

Dst.....

Penggalan program untuk grammar tsb



```
Prosedure Program
Begin
Ktoken=t_PROGRAM then
Begin
        Scan;
        If token=t_ID then
        Begin
                Scan;
                If token=t_SEMICOL then
                Begin
                        Scan;
                        Block; {panggil prosedur block}
                        If token=t_DOT then
                        Begin
                                Scan;
                        End;
                end;
        end;
end;
end;
```

```
Prosedure Block
Begin
If token=t_BEGIN then
begin
        scan;
        Statement; {panggil prosedur statement}
        While token=t_SEMICOL do
        Begin
                Scan;
                Statement;
        End;
        If token=t_END then
                scan
end;
end;
```