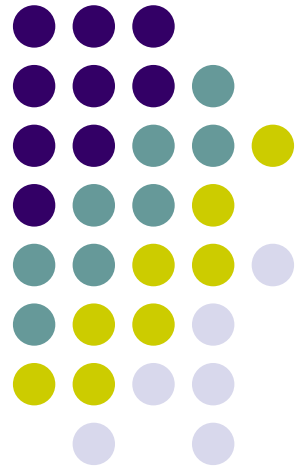
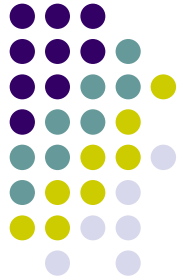
Teknik Kompiler 5

oleh: **antonius rachmat c, s.kom, m.cs**

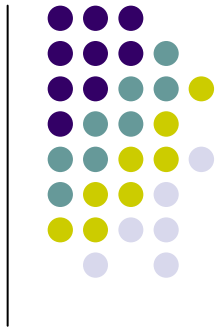


TATA BAHASA



- *Tata bahasa / Grammar* dalam OTOMATA adalah kumpulan dari himpunan variabel (non-terminal), simbol-simbol awal dan terminal yang dibatasi oleh aturan-aturan produksi.
- Grammar digunakan untuk mendefinisikan bahasa.
- Aturan produksi menspesifikasikan bagaimana suatu tata bahasa mentransformasikan suatu string ke bentuk lainnya. Biasanya aturan produksi diberi simbol:
  - **α → β**
    - dimana α adalah aturan produksi sebelah kiri
    - dan β adalah aturan produksi sebelah kanan
    - aturan produksi sebelah kir menghasilkan aturan produksi sebelah kanan.
- Simbol-simbol dalam aturan produksi dapat berupa simbol **terminal** maupun **non terminal**.

TATA BAHASA (2)



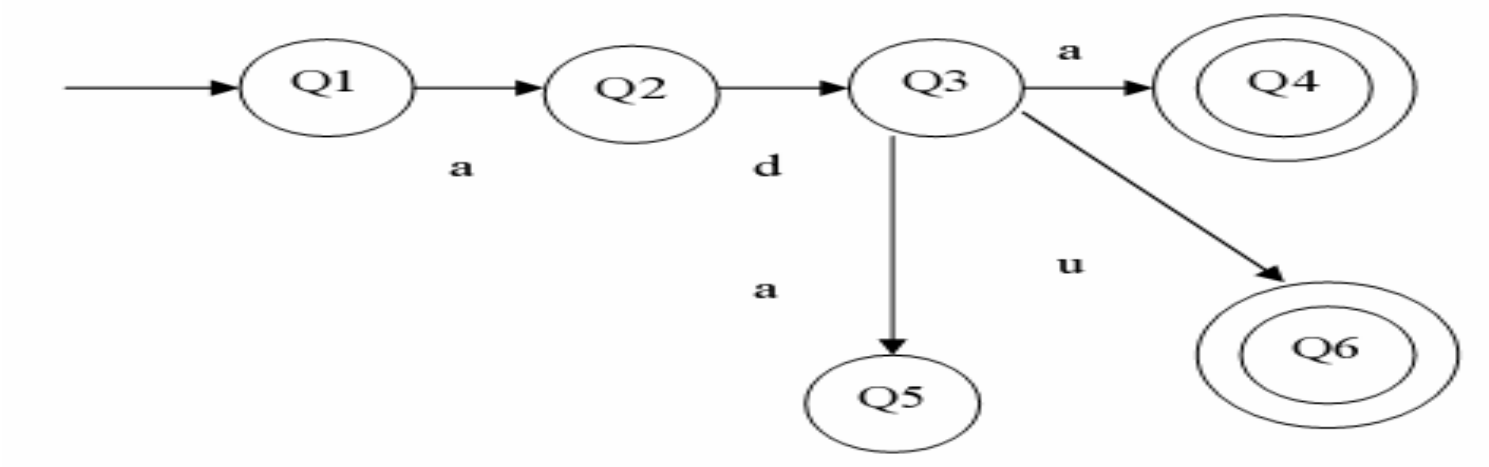
- Simbol terminal sudah **tidak dapat** diturunkan lagi.
- Simbol non-terminal **dapat** diturunkan lagi sampai menjadi simbol terminal.
- Simbol terminal biasanya ditulis dalam *huruf kecil*, sedangkan simbol non-terminal biasanya ditulis dalam *huruf besar*.
- Contoh simbol terminal : a,b,c,d,e, dan seterusnya.
- Contoh simbol non-terminal : A,B,C,D,E, dan seterusnya.

OTOMATA

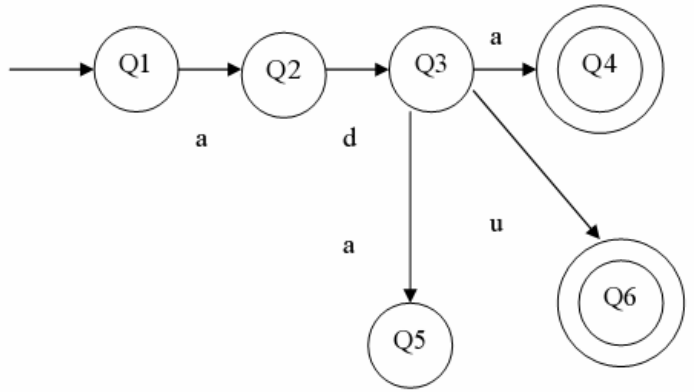
- Untuk memodelkan hardware dari komputer diperkenalkanlah *otomata / automata*. **Otomata** adalah suatu sistem yang memiliki fungsi-fungsi komputer digital.
- Karaketeristik Otomata :
  - Menerima input.
  - Menghasilkan output.
  - Mempunyai penyimpanan sementara (buffer)
  - Mampu membuat keputusan dalam mentransformasikan input ke output.

OTOMATA (2)

- Otomata merupakan suatu sistem yang terdiri atas sejumlah **berhingga** state, dimana setiap state menyatakan informasi tentang input sebelumnya, dan dapat dianggap sebagai memori mesin. Contoh:



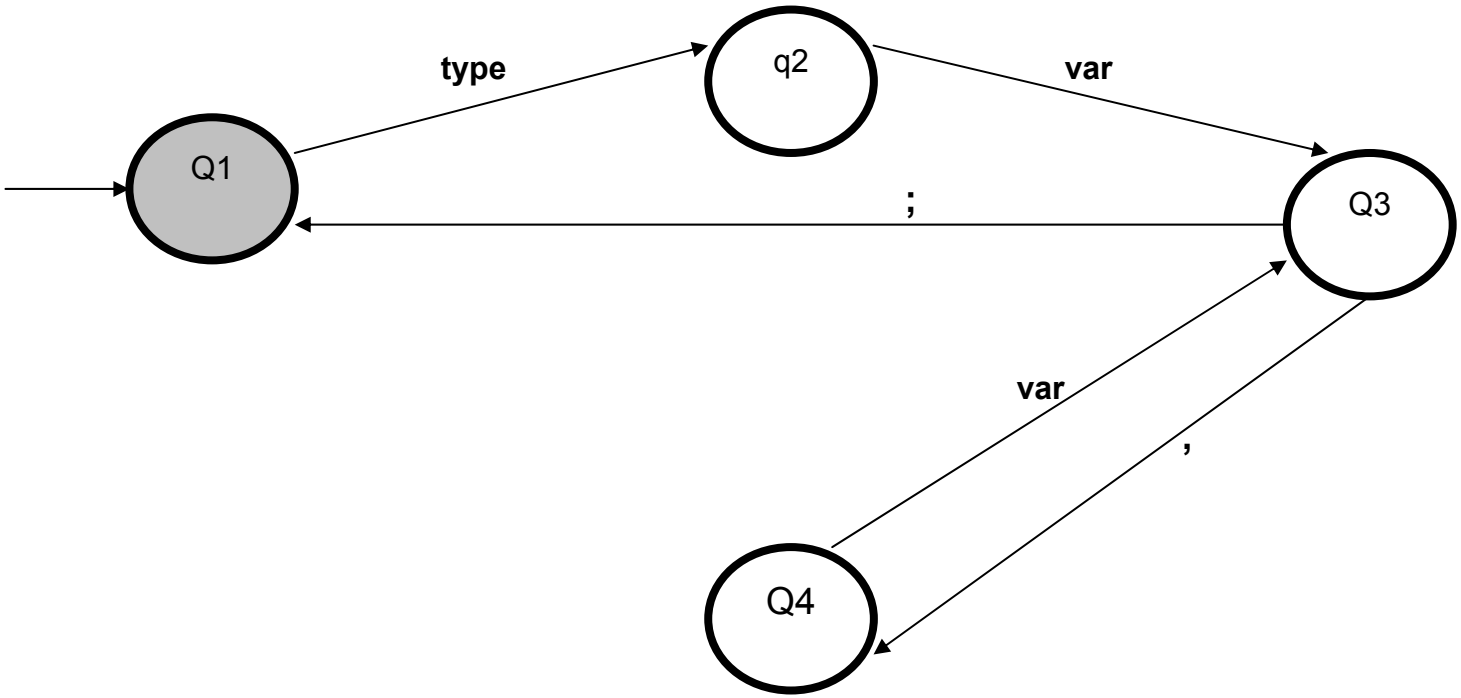
OTOMATA (3)



- Jika mesin mendapat string :
  - “ada” maka **diterima**
  - “adu” maka **diterima**
  - “add” maka **ditolak**
- Aturan: *input string diterima jika dan hanya jika mencapai state akhir yang disimbolkan dengan lingkaran ganda.*
- Keterangan:
  - Mesin diatas memiliki 6 state {q1,q2,q3,q4,q5,q6}.
  - Mesin memiliki state awal q1
  - Mesin memiliki state akhir {q4,q6}
  - Mesin memiliki himpunan input contohnya : {a,d,d}

Contoh Automata lain

- Contoh pada bahasa C: **int a,b;**



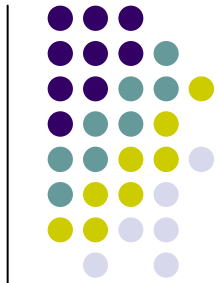
State awal: Q1, State akhir: Q1

Hirarki Chomsky

Pada tahun 1959 Noam Chomsky menggolongkan tingkatan bahasa menjadi empat yaitu:

Bahasa	Batasan Aturan Produksi
Natural Language (tipe 0)	Tidak ada batasan Contoh : Abc -> De
Context Sensitive (tipe 1)	|alpha| <= |beta| Contoh : Ab -> DeF ; CD -> eF ; S -> ε
Bebas Konteks (tipe 2)	alpha adalah sebuah simbol variabel Contoh : B –> CDdFg ; D -> BcDe
Regular (tipe 3)	alpha adalah sebuah simbol variabel dan beta maksimal memiliki sebuah simbol variabel yang terletak di posisi paling kanan. Contoh : A -> e ; A -> efg ; A -> efgH ; C -> D

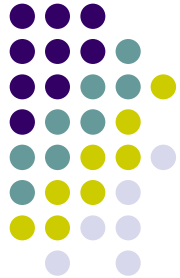
Hirarki Chomsky (2)



Catatan :

- ε → Abc adalah ilegal karena ε adalah himpunan kosong dan tidak bisa diturunkan lagi.
- a → bd atau ab → Bd adalah ilegal karena ruas kiri berupa simbol terminal, padahal seharusnya ruas simbol terminal sudah tidak bisa diturunkan lagi.

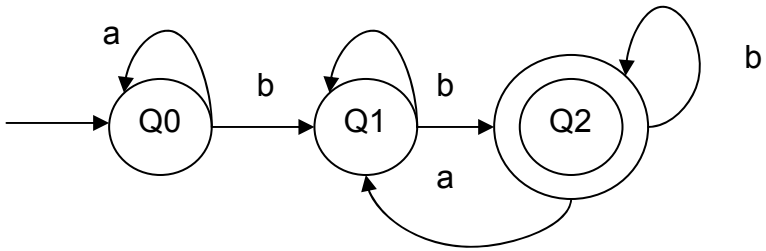
Finite State Automata (1)



- FSA adalah mesin otomata dari bahasa **regular**, memiliki state dengan jumlah berhingga dan dapat berpindah dari satu state ke state lain.
- Perubahan state ini dinyatakan dengan sebuah **fungsi transisi**.
- FSA tidak memiliki tempat penyimpanan sehingga kemampuan mengingatnya terbatas.
- FSA hanya mengingat kondisi (state) saat ini, satu state ke arah sebelum dan sesudahnya, dan sekumpulan perintah lain yang belum terpenuhi.
- Hanya bisa menggunakan bahasa tipe 3 (regular)
- Dibagi 2: FSA Deterministik & FSA non Deterministik

FSA Deterministik (DFA)

- Berarti : *setiap state memiliki tepat satu state berikutnya untuk setiap simbol masukkan yang diterima*.
- Setiap state selalu memiliki tepat satu state berikutnya.



Misalkan :
Himpunan state Q = {Q0,Q1,Q2}
Himpunan input ∑ = {a,b}
State awal S = Q0
State akhir F = Q2

Maka dapat digambarkan tabel transisinya sebagai berikut:

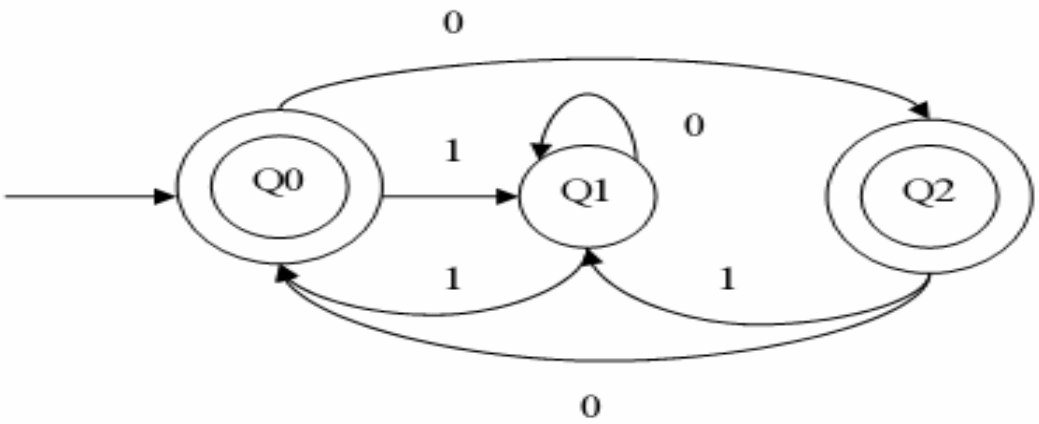
Fungsi	a	b
Q0	Q0	Q1
Q1	Q1	Q2
Q2	Q1	Q2

Contoh DFA

Jika terdapat sebuah tabel transisi

Fungsi	0	1
Q0	Q2	Q1
Q1	Q1	Q0
Q2	Q0	Q1

Maka bagaimana State diagramnya?

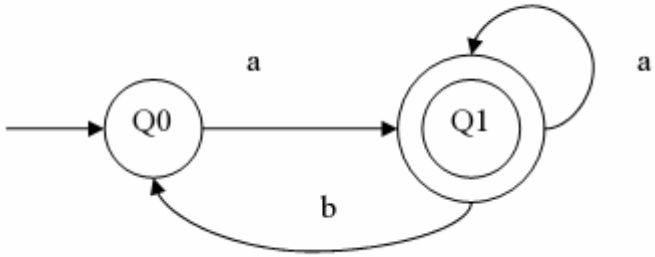


State awal: Q0
State akhir: Q0,Q2

FSA non Deterministik (NFA)

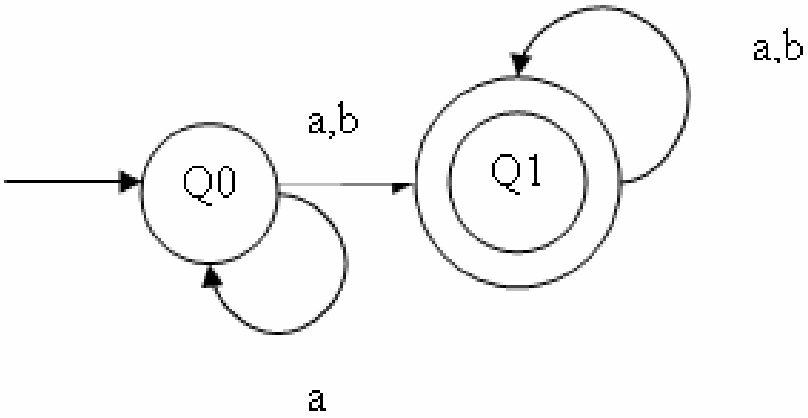
- Berarti : *Suatu state bisa memiliki 0,1 atau lebih output (panah) yang berlabel simbol input yang sama*.
- Digunakan tanda { } karena hasil transisi bisa berupa himpunan state

Contoh otomata yang bukan DFA (NFA) 1:



Fungsi	a	B
Q0	{Q1}	{}
Q1	{Q1}	{Q0}

Contoh NFA

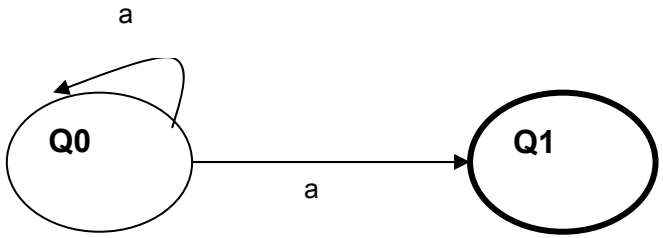


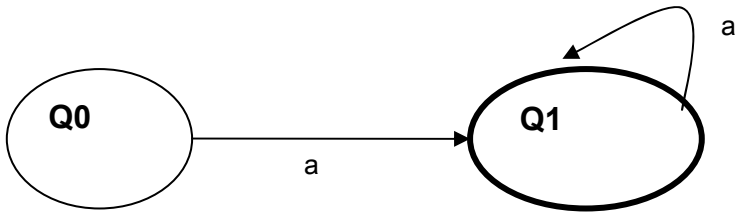
Tabel Transisinya

Fungsi	a	b
Q0	{Q0,Q1}	{Q1}
Q1	{Q1}	{Q1}

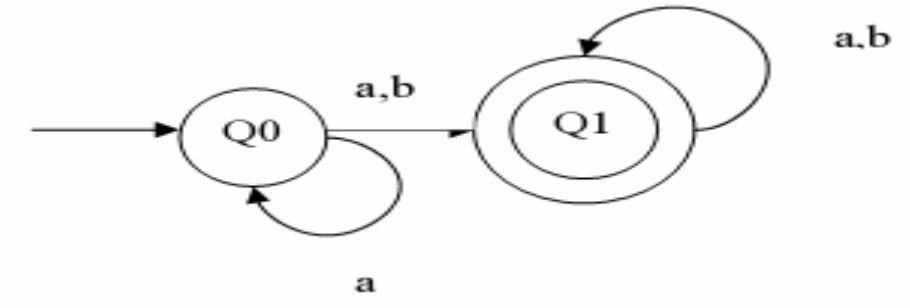
Ekuivalensi DFA

- Jika ada dua atau lebih DFA yang dapat menerima (accept) bahasa yang sama walaupun bentuk DFA nya berbeda, maka dua atau lebih DFA itu dianggap ekuivalen.

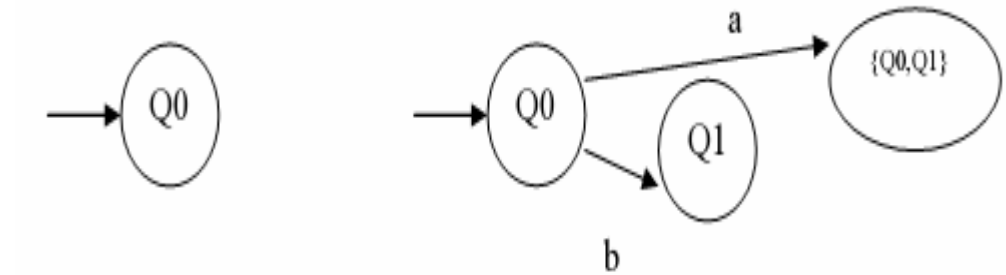




NFA bisa jadi DFA

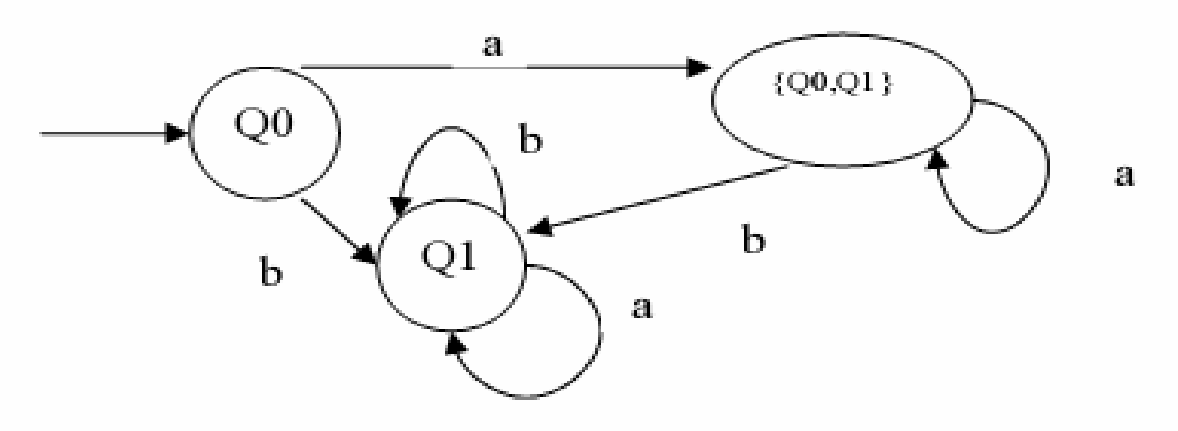


Lihat contoh NFA 2 diatas akan dijadikan DFA:



Mulai dari state awal Pilih state Q0

NFA -> DFA



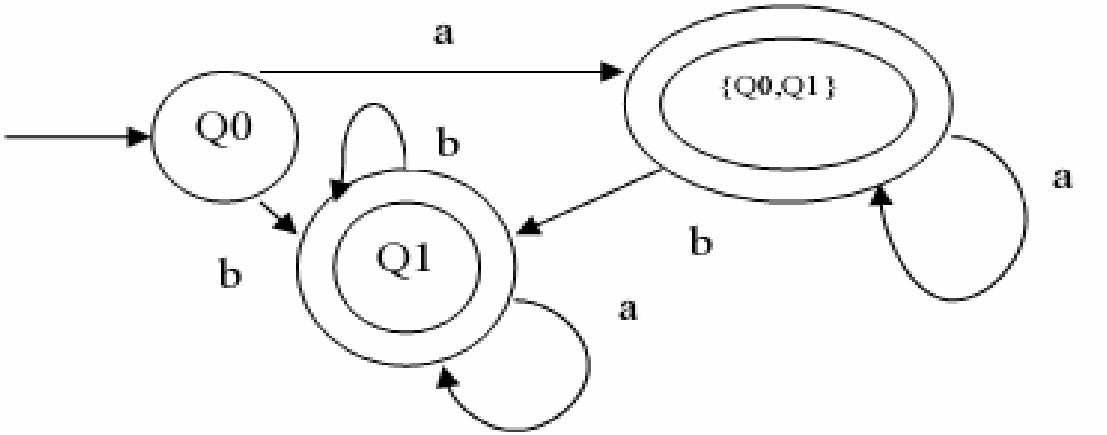
Pilih state Q1

Pilih state {Q0,Q1} :

F(Q0,a) = {Q0,Q1} digabung dengan F(Q1,a) = Q1 =>
{Q0,Q1} U {Q1} = {Q0,Q1}

F(Q0,b) = {Q1} digabung dengan F(Q1,b) = {Q1} =>
{Q1} U {Q1} = {Q1}

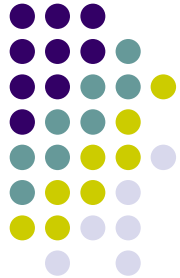
NFA -> DFA (2)



State akhir ada pada **Q1** berdasarkan pada NFA soal.
Jadi semua state yang mengandung Q1 menjadi state akhir. Maka sudah didapatkan bentuk DFA dari soal NFA diatas.
Masing-masing state telah memiliki tepat satu state keluar.

Pembuktian
Dari notasi NFA maupun DFA diatas, untuk string “aa” bisa diterima

Contoh NFA -> DFA lain

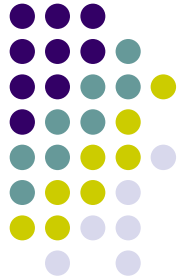


- Diagram State:

Current	Input a	Input b
1	{1,2}	{1}
2	{2}	{1,3}
3	{2,3}	{1,3}

Bagaimana diagramnya? Dan bagaimana NFA ke DFA nya?

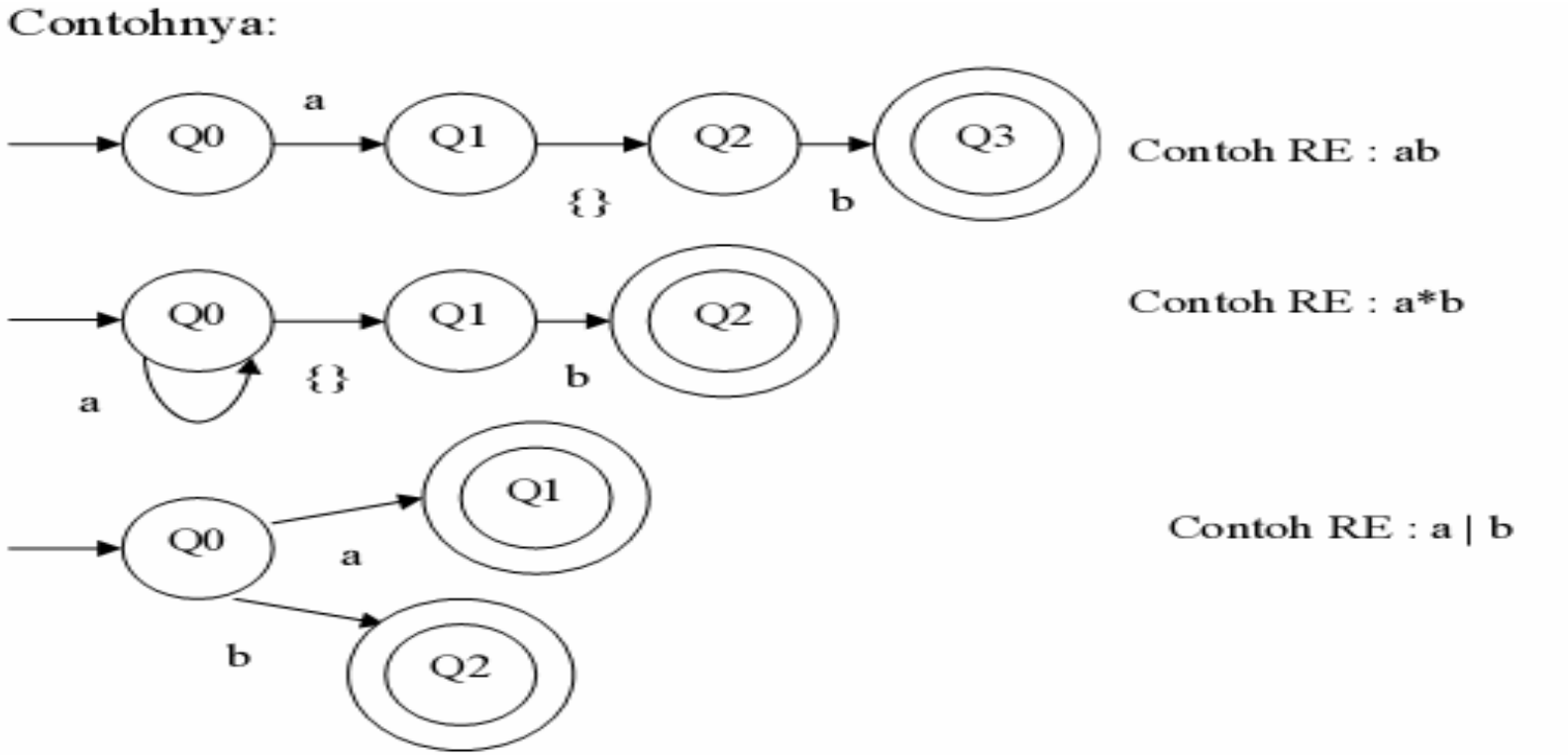
Pengembangan FSA



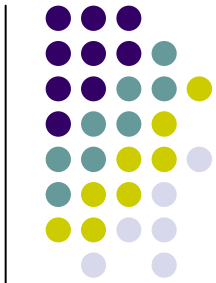
- **Recursive Transition Network**
  - FSA yang ditambah dengan kemampuan rekursif sehingga dapat mengenal tipe 2
  - Kemampuan node ditambah, sebuah node dapat berisi sub-network yang bisa diturunkan secara rekursif.
- **Augmented Transition Network**
  - RTN yang ditambah “register” yang menyimpan informasi.
  - Setiap panah dapat dieksekusi secara kondisional, dapat dilakukan test terlebih dahulu.
  - Menambahkan kemampuan “action” ke panah, dalam bentuk pengubahan struktur data yang dipakai.
  - Contoh: NLP programs use it!

RE

Contoh penerapan DFA ataupun NFA adalah pada *Regular Expression (RE)* seperti yang telah kita bahas minggu lalu.



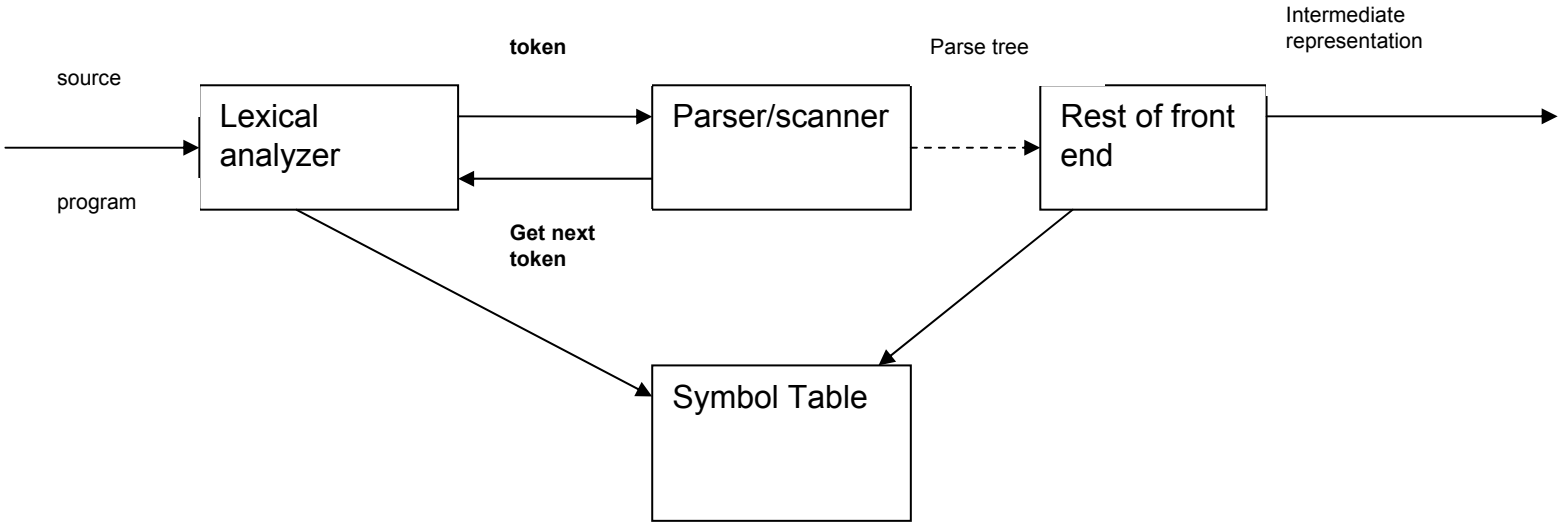
Analisis Lexical (Scanner)



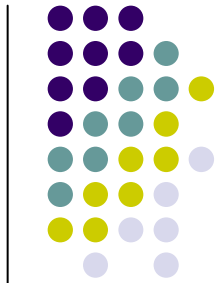
- Scanner berfungsi melakukan analisis leksikal, yaitu mengidentifikasi semua “simbol” yang membangun suatu bahasa pada suatu source code.
- Scanner bertugas:
  - Membuang komentar
  - Menyeragamkan huruf (menjadi besar atau kecil) pada semua identifier pada bahasa pemrograman yg incasesensitive
  - Membuang white space
  - Mengintepretasikan kompiler directive (misal: {$R+, {$R-, include, dll)
  - Berkomunikasi dengan tabel simbol

Token, Lexem, Diagram

- Lexem adalah substring dari suatu string besar, menjadi satu bagian tertentu yang memiliki arti
- Token adalah suatu nama (identifier) yang sudah didefinisikan fungsionalitasnya dan merupakan kumpulan dari lexem-lexem.

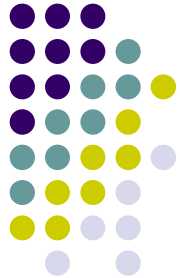


Token & Lexem



- Contoh non-token:
  - White space character : space, tab, end-of-line
  - Comments
- Contoh input :
  - If distance >= rate*(end_time – start_time) then distance:= maxdist;
- Output Lexem :
  - if, distance, >= , rate, *, (, end_time, - , start_time,), then, distance, :=, maxdist, ;
- Output Token :
  - If_op, id_var (distance,rate,end_time, start_time, maxdist), gt_op (>=), optr (*, -), assig_op (:=), block_op ((,)), then_op, end_op

Scanner

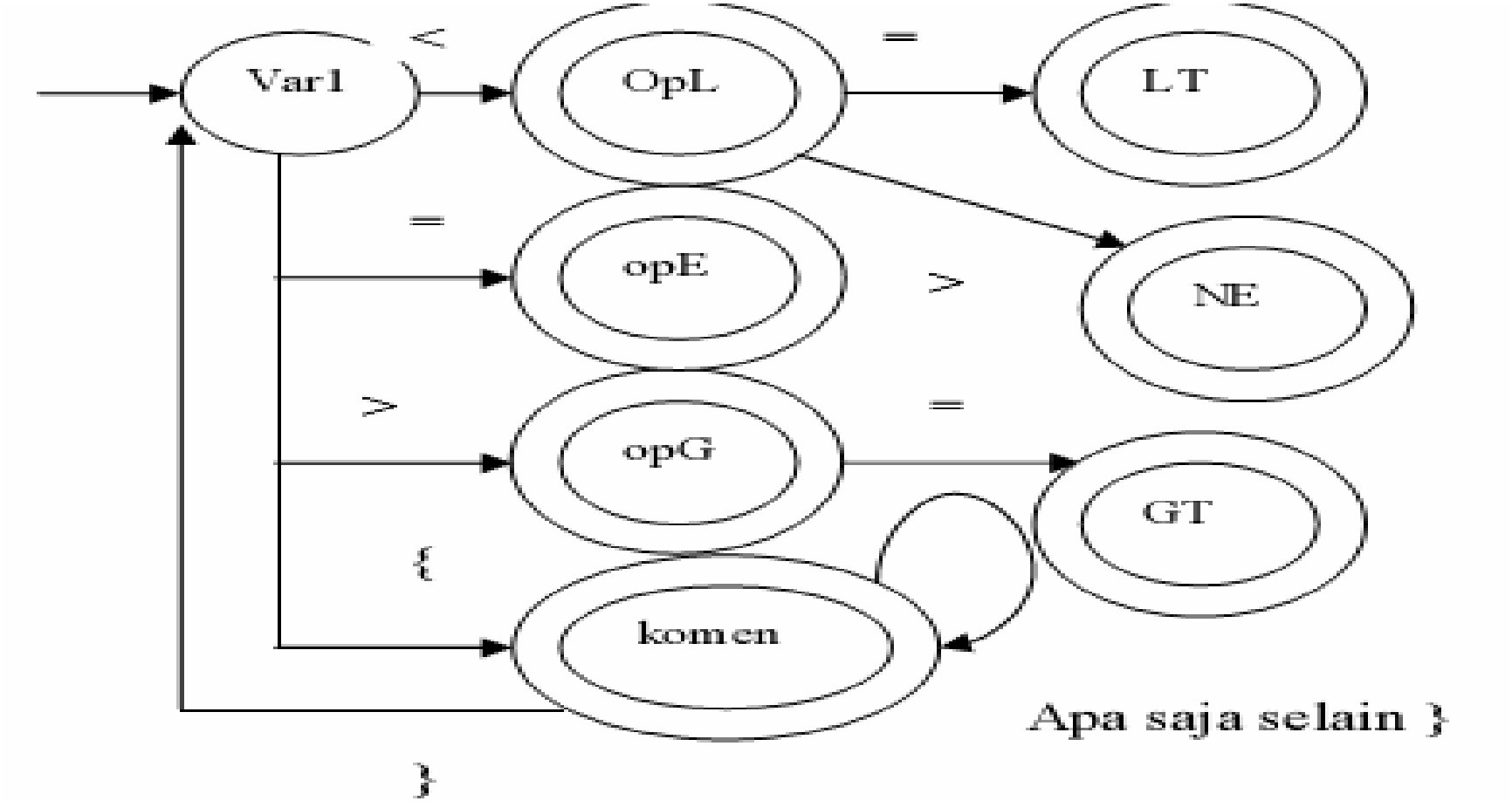


- Token dan source code akan menjadi input bagi proses berikutnya, yaitu parser.
- Parser akan menerima hasil analisis lexical dalam bentuk:
  - If_op id_var1 gt_op id_var2 optr_1 block_op1 id_var2 optr_2 id_var3 block_op2 then_op id_var3 assig_op id_var3 end_op
- Scanner lebih mudah diimplementasikan pada Finite State Automata dan aturan-aturan tokennya ditulis dengan menggunakan RE.

Lexem, Operator, dan Delimiter

- Jenis-jenis Lexem:
  - Identifier
    - Bisa berupa keyword atau nama variabel.
    - Keywords adalah kata-kata yang sudah didefinisikan oleh program, sedangkan variabel adalah kata-kata yang didefinisikan oleh programmer.
  - Nilai konstanta
    - Adalah nilai yang bersifat konstan. Dapat berupa integer, real, boolean, karakter, ataupun string. Contoh: N := 0.333; kata := ‘malam’; selesai := TRUE; A := 1 * 2 + 3;
  - Operator dan Delimitter
    - Operator : + , - , * , / , < , <= , >=, >, = dan lain-lain.
    - Delimiter : (,) , ; , dan lain-lain.

RE untuk operator dan komentar pd Pascal

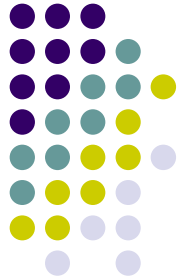


```
var fileinput : text;
      kar : karakter;

procedure getchar;
begin
      Read(fileinput,kar);
end;

procedure scan;
begin
      while kar = ' ' do
            getchar;
      repeat
            case kar of
            '<'   : begin
                        getchar;
                        case kar of
                        '='   : begin token := 'opLT'; exit; end;
                        '>' : begin token := 'opNE'; exit; end;
                        else  begin token := 'opL'; exit; end;
                      end;
            '='   : begin
                        token := 'opE'; exit; end;
                      end;
            '>'   : begin
                        getchar;
                        if kar = '=' then begin token := 'opGT'; exit;
end;
                        else  begin token := 'opG'; exit; end;
                      end;
            '{'   : begin
                        repeat
                              getchar;
                        until kar='}';
                        getchar;  //Lanjutkan  maju,  tanpa  memperoleh
token
                      end;
            EOF   : exit; //akhir file
      until false;
end;
```

NEXT



- See you on TTS!
- Sifat: buka selembar kertas (bolak-balik, boleh diketik)
- Bahan: dari awal hingga materi ini
- Senin, 14 juli 2008, pukul 7:30 (70 menit), D11
- Soal: pilihan ganda dan essay!
- Do it yourself and GBU!